MGM-AE: Self-Supervised Learning on 3D Shape Using Mesh Graph Masked Autoencoders

Zhangsihao Yang
Arizona State University
zshyang1106@gmail.com

Kaize Ding
Northwestern University
kaize.ding@northwestern.edu

Huan Liu
Arizona State University
huanliu@asu.edu

Yalin Wang
Arizona State University
ylwang@asu.edu

Abstract

The challenges of applying self-supervised learning to 3D mesh data include difficulties in explicitly modeling and leveraging geometric topology information and designing appropriate pretext tasks and augmentation methods for irregular mesh topology. In this paper, we propose a novel approach for pre-training models on large-scale, unlabeled datasets using graph masking on a mesh graph composed of faces. Our method, Mesh Graph Masked Autoencoders (MGM-AE), utilizes masked autoencoding to pre-train the model and extract important features from the data. Our pre-trained model outperforms prior state-of-the-art mesh encoders in shape classification and segmentation benchmarks, achieving 90.8% accuracy on ModelNet40 and 78.5 mIoU on ShapeNet. The best performance is obtained when the model is trained and evaluated under different masking ratios. Our approach demonstrates effectiveness in pre-training models on large-scale, unlabeled datasets and its potential for improving performance on downstream tasks.

1. Introduction

Mesh is a widely-used data format in computer graphics and has become a prevailing format for capturing continuous underlying surfaces due to its capability of providing an accurate, efficient, and irregular representation of three-dimensional shapes. Many commonly used compute vision datasets, such as ModelNet [63], ShapeNet [7], ScanNet [9], and Pix3D [51], utilize meshes as the core or intermediate representation format. Deep learning on meshes has applications of classification and segmentation [20], generation [17], and animation [41].

Due to the fact that data labeling on 3D shape is labor-intensive and resource-expensive, self-supervised learning has emerged as a powerful technique for training machine learning models using unlabeled data, either in a generative way [1, 22] or contrastive way [2, 6, 12, 18, 27, 42, 44, 58–60]. Benefiting from the capability of new model architecture [56], self-supervised learning has demonstrated state-of-the-art performance in various domains, such as image [43], video [61], and text [5]. More recently, self-supervised representation learning has also been applied to 3D imaging, primarily point cloud [23] and 3D voxel grids [39].

Despite the fruitful success that has been made in self-supervised learning for 3D imaging, there have been limited works that successfully apply self-supervised learning to 3D mesh. This research problem remains to be non-trivial mainly due to the following challenges: (1) on the one hand, the feature extractors of existing works are commonly designed for those data formats derived from mesh (e.g., point cloud), which inevitably lose the inherent geometric topology information within the mesh data to some extent. Though recent work [36] proposes to extract patches from the mesh and then feed them into transformers for learning the mesh representations. In their approach, both a fixed patch topology and a predetermined number of patches are mandated, meanwhile the Transformer is based on global self-attention that disregards the local connectivity of meshes. Hence, how to explicitly model and leverage the inherent geometric topology information is a key to learning expressive representations on 3D mesh data; (2) on the other hand, due to the irregular topology of 3D mesh, directly applying existing self-supervised learning strategies for other 3D data formats may easily lose their efficacy on meshes. It remains unclear how to design appropriate pretext tasks including both the augmentation method and self-supervised learning objective on 3D mesh data for better exploiting the geometric topology information.

In this paper, we propose to solve the 3D mesh self-supervised learning problem from a graph learning perspec-
tive. Specifically, we treat each face of the mesh as a node and build a mesh graph to model each 3D shape. To learn the 3D shape representations without any semantic labels, we innovatively develop a mesh-based self-supervised learning framework, Mesh Graph Masked Autoencoder (MGM-AE), which can be pre-trained on large-scale 3D imaging datasets. Different from previous works [20, 57] that mimic regular convolution to meshes, the encoder of MGM-AE adopts the graph attention layer as the building block, which is able to explicitly capture the irregular topology knowledge of mesh graphs while attentionally considering the importance of each node (face) during message-passing. In order to improve the expressiveness of learned 3D mesh representations, we follow the recent advances in self-supervised learning [22] and design a masked autoencoding pretext task on the mesh graphs. Specifically, we randomly replace certain nodes’ features with masked features and perform node feature reconstruction based on the representations learned from the graph attention encoder. By the virtue of graph attention encoder, the representation of each node on a mesh not only encodes the information from its corresponding face, but also captures the information from its neighboring faces via multi-hop message passing. This way allows the perturbed 3D meshes can be more effectively reconstructed even with a lightweight decoder (i.e., MLP) during the decoding phase, such that expressive representations for 3D meshes can be learned. Our simple yet effective framework is compatible with size-varying meshes, which increases the model flexibility for dealing with a variety of datasets. To demonstrate the effectiveness of our method, we perform a variety of experiments and show state-of-the-art performance on different downstream tasks compared to other mesh-based shape feature extractors. To summarize, the key contributions of our work are as follows:

- We propose to solve the problem of self-supervised learning on 3D mesh data from a graph learning perspective, which goes beyond the existing paradigm and sheds light on the following research.

- We introduce a Mesh Graph Masked Autoencoder (MGM-AE), a novel mesh-based masked encoding pretraining framework that leverages the inherent topology information of mesh data, thereby enhancing the expressiveness of mesh representations.

- Our comprehensive evaluations on various benchmarks such as SHREC11, ModelNet40, and ShapeNetPart, demonstrate that our MGM-AE model outperforms prior mesh-based neural network models and achieves state-of-the-art performance in supervised and self-supervised classification as well as semi-supervised segmentation tasks.

2. Related Work

Deep Learning on Meshes In general, deep learning on polygonal meshes can be summarized in two main categories: (1) graph-based methods, and (2) manifold-based methods. Graph-based methods try to process the mesh data directly by extracting locally connected regions and converting them into a graph form for subsequent graph neural networks learning. For example, FeaStNet [57] proposes a graphical neural network in which the neighborhood of each peak for the convolution operation is not preset but instead calculated dynamically. MeshCNN [20] utilizes the particular property of edge in a triangle mesh to extract edge features. Subsequent works, such as [10, 35], that build upon MeshCNN [20], similarly treat edges as nodes in a graph. Specifically, FPCNN [35] opts for quadric error metrics over learning-based pooling [20]. MEAN [10] introduces edge attention to enhance edge-based graph convolution. Bending Graphs [47] use graph neural networks to incorporate local and global graph information for shape matching problems. Our backbone model also draws inspiration from the aforementioned methods but treats faces as nodes in a graph, which better captures the manifold nature of meshes.

Most non-graph approaches treat meshes as manifolds and develop methods to adapt convolution operations from structured grids, like images, to unstructured ones, such as meshes. Geodesic CNN [37], MoNet [40], and SplineCNN [14] deal with the weight sharing problem by designing local coordinate systems for the central vertex in a local patch. Those methods apply a set of weighting functions to aggregate the characteristics at the adjacent vertices and then calculate a weighted mean of aggregated information. However, these methods are informatically expensive and require pre-defined local coordinate systems. [54] first proposes a transformer-based procedure for the efficient registration of non-rigid 3D point clouds. Neural3DMM [4] uses a spiral convolution to order vertices based on the shortest geodesic path to a template reference. However, selecting a reference for arbitrary shapes is challenging, and ambiguity arises when adjacent vertices share the same path length. Tangent convolution is introduced in [53], where a small neighborhood around each vertex is utilized to reconstruct the local function, upon which the convolution is applied. Cont-Conv [65] employs continuous convolution over a geodesic region of the mesh. However, determining a local coordinate system and projecting neighbors onto this local system can be ambiguous and computationally expensive. The recently proposed MeshMAE method [36] splits a mesh into patches, each with a fixed topology, and feeds a fixed number of patches into the Transformer [56]. Their method may lose information during the remeshing of complex topology meshes and highly relies on hand-crafted patch features to be sufficiently informative. In contrast, our method emphasizes attention among adjacent mesh faces, enabling it to handle
intricate topologies without information loss and obviating the need for hand-crafted patch features as the fundamental unit of analysis.

**Self-Supervised Learning** Self-supervised learning involves defining pretext tasks directly from the data, using these human-defined tasks to pre-train the model. It is used in computer vision with pretext tasks such as predicting order in time [60], finding missing pixels [44], location of patches [12], image orientations [18], human-made artifacts [27], clusters of images [6], camera locations [2], jaggle puzzle [42], color of videos [58], and tracking of image patches [59]. These works demonstrate promising results in transferring visual features from pretext tasks to other tasks. Thus, defining pretext tasks that are related enough to the downstream task is quite important [27].

Studies exploring self-supervised learning on 3D data have been centered around point clouds. They use multi-task learning [21], reconstruction [1], contrast learning [67], restoring point cloud [48], point cloud autoregression [52], the orientation prediction [19], and approximating convex decomposition [15] to pre-train the model and achieve state-of-the-art results on point cloud classification and segmentation tasks. While there is an abundance of research on self-supervised learning for point clouds, studies focusing on meshes are limited. Recently, masked autoencoders, which restore data from masked input, have gained traction in self-supervised learning, particularly for images [22] and graphs [68]. MeshMAE [36] integrates masked autoencoders with meshes from a manifold perspective.

**3. Method**

MGM-AE is a masked autoencoder that interprets the mesh as a graph, and each graph node is a face on the mesh. The features on the face nodes are randomly masked first and passed through multiple face graph attention layers. Then max-pooling is applied to obtain the global graph embedding, which is passed to a point cloud decoder for reconstruction pre-training tasks.

**Face descriptor** is designed in our method to represent faces in meshes. For face \( i \) in our graph, we include the center of the face \( c_i \in \mathbb{R}^3 \), the normal direction of the face \( n_i \in \mathbb{R}^3 \), the radius of the circle covering the triangle \( r_i \in \mathbb{R} \), and normalized directions \( v_1, v_2, v_3 \in \mathbb{R}^3 \) from the center to three vertices sorted according to the degree of the inner angle. These geometric features are sent through four linear layers and concatenated together as the face descriptor.

\[
\mathbf{h}_i^0 = f_c(c_i) \parallel f_n(n_i) \parallel f_r(r_i) \parallel f_v(v_1) \parallel f_v(v_2) \parallel f_v(v_3) \quad (1)
\]

In our experiment, we set \( f_c : \mathbb{R}^3 \rightarrow \mathbb{R}^6 \), \( f_n : \mathbb{R}^3 \rightarrow \mathbb{R}^6 \), \( f_r : \mathbb{R} \rightarrow \mathbb{R}^6 \), \( f_v : \mathbb{R}^3 \rightarrow \mathbb{R}^6 \) which is shared across \( v_1, v_2, v_3 \). And input to the face graph attention layer is \( \mathbf{h}_i^0 = \{ \mathbf{h}_i^0, \mathbf{h}_i^0, ..., \mathbf{h}_i^0 \} \in \mathbb{R}^{N \times 36} \) where \( N \) is the number of nodes in the graph.

**Masking on face graph** is achieved by randomly selecting nodes on the graph according to the masking ratio. After one node is selected as the masked node, a learnable
masking embedding $h^k_m \in \mathbb{R}^{36}$ takes the place of the original embedding, which is extended from \cite{11, 22}.

**Face graph attention layer** is the core of our network, as shown in Figure 1. We take the $k$th layer as an example and the architecture is composed of duplicates of the layer. The $k$th layer takes a graph and the features $h^{k-1} \in \mathbb{R}^{N \times d^{k-1}}$ of the graph as input and outputs $h^{k} \in \mathbb{R}^{N \times d^{k}}$ where $d^{k-1}$ and $d^{k}$ are the input and output dimension of the $k$th layer. For node $i$ in the graph, the layer first gathers its neighbors $\mathcal{N}_i$ according to an adjacency matrix which could be an n-ring neighbor adjacency matrix. We denote $h^{k-1}_i$ as the input feature of the face node $i$ and $H^{k-1}_i = \{h^{k-1}_j : j \in \mathcal{N}_i\} \in \mathbb{R}^{N_i \times d^{k-1}}$, where $N_i$ is the number of neighboring nodes of node $i$, as the gathered neighboring features of the root node. Three linear layers $f_v$, $f_q$, and $f_k$ take $h^{k-1}_i$, $H^{k-1}_i$, and $h^{k-1}_i$ as input to compute value $V^k_i \in \mathbb{R}^{1 \times d^k}$, query $Q^k_i \in \mathbb{R}^{N_i \times d^k}$, and key $K^k_i \in \mathbb{R}^{1 \times d^k}$ where $d^k$, $d^k$, and $d^k$ are the dimensions of the output with $d^k = d^k$ and $d^k = d^k$. We expand one extra dimension in $h^{k-1}$ in order to explain equation 2. In our work, we keep $d^k$, $d^k$, and $d^k$ fixed to 64.

$$h^k = \text{softmax} \left( \frac{Q^k_i K^k_i^T}{\sqrt{d^k}} V^k_i \right) \quad (2)$$

We use Equation 2 to get the output feature $h^k_i$ of face node $i$ and the output of the $k$th layer is $h^k = \{h^k_1, h^k_2, ..., h^k_N\}$. Details of composing the layers into an encoder are in Section 2 in the appendix.

**Reconstruction loss** In the reconstruction loss function, a reconstruction decoder is utilized. The input to this decoder is the graph embedding of the mesh. The expected output is the point cloud sampled from the mesh. Following \cite{1}, we use a similar network architecture $f_d$ for decoding a point cloud. So we choose the point cloud as the target for the decoder to generate. And the loss function is the Chamfer Distance (CD), as shown in Equation 3.

$$\mathcal{L}_{CD} = \frac{1}{N} \sum_{n=1}^{N} \min_{\hat{s} \in \hat{s}} \|p_n - \hat{s}\|^2_2 + \frac{1}{M} \sum_{m=1}^{M} \min_{\hat{s} \in \hat{s}} \|\hat{p}_m - s\|^2_2 \quad (3)$$

where $s$ and $\hat{s}$ are the ground truth and predicted point sets. $M$ and $N$ denote the number of points in the ground truth and predicted point sets. $p_n$ and $\hat{p}_m$ are points sampled from point set $s$ and $\hat{s}$.

### 4. Experiments and Results

In this section, we introduce experiments to validate the effectiveness of our neural networks. First, we demonstrate the effectiveness of the encoder part of our networks on two supervised classification tasks. Then, we verify our work by pre-training the network on an unsupervised classification task and transferring the learned features for supervised classification. Finally, we conduct a semi-supervised experiment for part segmentation on 3D shapes.

#### 4.1. Supervised Classification

We first verify that our network’s encoder could outperform other networks. By using the designed mesh graph attention encoder, we achieve state-of-the-art performance on SHREC11 and ModelNet40 with mesh inputs.

**SHREC11** is a dataset introduced in \cite{34} that contains 30 classes, with 20 3D objects in each class. We follow setups in \cite{20} which split 16 and 10 are the numbers of training 3D objects in each class, making split 10 a harder classification task than split 16. We use the meshes processed by \cite{20} and each mesh contains 500 faces. Our results are reported in Table 1. We train our encoder 300 epochs with Adam optimizer, \cite{30} which is with $\beta$ equal to 0.9 and 0.999, $\epsilon$ equal to $1^{-8}$, learning rate 0.0002 and weight decay equal to 0.0. We compare our mesh graph attention encoder against eight methods that also take meshes as the input to their networks. It turns out that our encoder is able to get 100% accuracy on both setups.

Because SHREC11 is a relatively small dataset for supervised classification and some methods have reached 100%
Figure 2. On the left, we show that the model converges faster when training from the pre-trained model. On the right, we visualize the Hinge loss landscape of the pre-trained model and randomly initialized model when the classification head is an SVM.

accuracy, we further validate our mesh graph attention encoder on ModelNet40 [63].

ModelNet40 is a dataset that contains 40 classes, and there are 9840 meshes for training and 2468 meshes for testing. Because meshes in ModelNet40 have different numbers of faces. To fit meshes onto GPU and to improve the GPU utilization, we follow the method in [26] to first make the mesh watertight, then simplify the meshes into 2048 faces. We train our encoder 300 epochs with the same optimizer settings as for SHREC11. The learning rate is decayed by a multiplicative factor of 0.1 at steps 30 and 60. Our method achieved 93.0% test accuracy on ModelNet40.

The results are reported in Table 2. We compare our encoder with six mesh-based methods. Our results are on par with state-of-the-art classification on ModelNet40 when training the model from scratch. When we fine-tune the model using our proposed mesh auto-encoding algorithm, the results indicate that our pre-trained algorithm is capable of providing superior starting points. We achieve an impressive 93.2%. It’s important to note that we utilize the same data for both pre-training and supervised training to ensure a fair comparison between these two methods.

In Figure 2, we present two key observations. Firstly, our pre-trained model not only converges more rapidly but also settles at a relatively lower local minimum. Secondly, as depicted on the right side of the figure, our pre-trained model offers a decidedly better starting point for searching compared to a model with randomly initialized weights. More specifically, on the right side of Figure 2, we employ the method from [33] to sample weights along the pre-trained weights and randomly initialized weights. We then calculate the Hinge Loss as the y-value to illustrate the landscape of these two model weights. This visualization implies two things: Firstly, our model offers a lower starting point, corroborating the fact that our pre-trained model converges more swiftly. Secondly, our model provides a flatter starting point, which is recognized to have superior generalization as mentioned in [24].

These experiments validate that our encoder could get state-of-the-art performances on 3D shape classification tasks. The next experiments are to validate the model’s pre-training performance on downstream tasks.

<table>
<thead>
<tr>
<th>Method</th>
<th>Modality</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>LGAN [1]</td>
<td>Point</td>
<td>84.5</td>
</tr>
<tr>
<td>MRTNet [16]</td>
<td>Point</td>
<td>86.4</td>
</tr>
<tr>
<td>PCGAN [32]</td>
<td>Point</td>
<td>87.8</td>
</tr>
<tr>
<td>FoldingNet [64]</td>
<td>Point</td>
<td>88.4</td>
</tr>
<tr>
<td>PointGrow [52]</td>
<td>Point</td>
<td>85.8</td>
</tr>
<tr>
<td>NSampler [46]</td>
<td>Point</td>
<td>88.7</td>
</tr>
<tr>
<td>3D-PointCapsNet [71]</td>
<td>Point</td>
<td>88.9</td>
</tr>
<tr>
<td>Multi-task [21]</td>
<td>Point</td>
<td>89.1</td>
</tr>
<tr>
<td>PointDist [48]</td>
<td>Point</td>
<td>84.7</td>
</tr>
<tr>
<td>ACD [15]</td>
<td>Point</td>
<td>89.8</td>
</tr>
<tr>
<td>PointOE [45]</td>
<td>Point</td>
<td>90.8</td>
</tr>
<tr>
<td>PTv1 [70]</td>
<td>Point</td>
<td>84.6</td>
</tr>
<tr>
<td>GSIR [8]</td>
<td>Point</td>
<td>90.4</td>
</tr>
<tr>
<td>MAP-VAE [19]</td>
<td>Point</td>
<td>90.2</td>
</tr>
<tr>
<td>PTv2 [62]</td>
<td>Point</td>
<td>86.3</td>
</tr>
<tr>
<td>ContrastNet [67]</td>
<td>Voxel</td>
<td>86.8</td>
</tr>
<tr>
<td>AnyPoint [69]</td>
<td>Points+Voxel</td>
<td>86.4</td>
</tr>
<tr>
<td>SPH [28]</td>
<td>Mesh</td>
<td>68.2</td>
</tr>
<tr>
<td>FeaStNet [57]</td>
<td>Mesh</td>
<td>74.4</td>
</tr>
<tr>
<td>MeshCNN [20]</td>
<td>Mesh</td>
<td>76.8</td>
</tr>
<tr>
<td>ContConv [65]</td>
<td>Mesh</td>
<td>76.5</td>
</tr>
<tr>
<td>MeshMAE [36]</td>
<td>Mesh</td>
<td>89.2</td>
</tr>
<tr>
<td>MGM-AE(Ours)</td>
<td>Mesh</td>
<td>90.8</td>
</tr>
</tbody>
</table>

Table 3. Accuracy of transfer learning methods for classification on ModelNet40. We compare multiple methods taking different modalities of 3D data, including point cloud, voxel, and mesh.

4.2. Transfer Learning for Classification

We process all the provided training data (57000 models across 55 categories) in ShapeNet [7] in the same way as ModelNet40 and pre-train the model on the data. We keep the pre-trained model’s weight and use it for classification tasks. We do not perform fine-tuning when using the pre-trained model for downstream tasks. After obtaining the graph embedding, we use a linear Support Vector Machine (SVM) as the classification tool for classification on ModelNet40.

The process of our self-supervised learning is stated as follows. We first pre-train the masked autoencoder with training data with the same training hype-parameter setting as in Section 4.1. After pre-training the model, we pick the model with the lowest Chamfer Distance on provided validation data in ShapeNet. We use the best model to extract global embeddings from the training and test data in ModelNet40, a vector with dimension 1024. Once we obtain the global embeddings, we use linear SVMs to train on Mod-
elNet40 training data’s global embeddings. We use 5-fold cross-validation to compute the average validation accuracy on the data split from training data. We also perform a logarithm search on the regularization parameter $C$ of SVM from 1 to 1000 with the number of steps equal to 10. Then we pick the SVM model with the best average validation accuracy to compute the test accuracy. In Section 1 in the appendix, we visualize graph embeddings with t-SNE [55].

In Table 3, our method outperforms other mesh-based neural networks on self-supervised pre-training on ModelNet40. There are two reasons our method outperforms other mesh-based methods. The first reason is our encoder utilizes an attention mechanism to pick important points while ignoring the noisy information by assigning lower weights to the noisy neighboring. The second reason could contribute to the masking mechanism. It provides more data augmentation to our model and forces the model to focus less on the details of the shapes than on the general information in the graph. And [45] that performs on par with our methods is a point cloud-based method. The possible reason could be that data augmentation, like rotation [19], is not considered when designing our framework. Adding rotation-invariant or equivariant design components to our framework is worth exploring in future work. For Point Transformers [62, 70], we leverage their pretrained weights on S3DIS [3]. Subsequently, we train a linear SVM to classify the point cloud, utilizing the features extracted from these pretrained weights.

In Figure 3, we show the reconstruction results on ModelNet40 test data. To some extent, the autoencoder ignores the input mesh’s detailed features while preserving the input mesh’s overall structure. Those detailed features, like the airplane’s engine, the chair’s arm, and the leg style of a table, are ignored during the reconstruction. Ignoring those detailed features means that the encoder encodes the information that is good for decoding into an average shape in the class but forgets details. For reconstruction tasks, this is not desired. But for classification, this process is like cleaning redundant information from the input shape. More reconstruction results are in Figure 5 in the appendix.

4.3. Part Segmentation

Part segmentation is a fine-grained point-wise classification task that aims to predict each point’s part category label in a given shape. In our work, we need to predict the part category label for each face in a mesh. We evaluate the learned point features on the ShapeNetPart dataset [66], which contains 16,881 objects from 16 categories (12149 for training, 2874 for testing, and 1858 for validation). Each object consists of 2 to 6 parts with a total of 50 distinct parts among all categories. We use the mean Intersection-over-Union (mIoU) as the measurement calculated by averaging the IoUs of the different parts occurring in one shape.

For the segmentation result, we follow the protocol from [21]. The results are shown in Table 4. In the original dataset, only point clouds and their corresponding point-wise labels are provided. To get ground truth for meshes, we need to first align the mesh with the point cloud by sampling points on the mesh and align the centers of the sampled point clouds with the provided point clouds. After the alignments, we first sample points on the face uniformly for each face on the mesh. Then we compute the nearest point in the ground truth point cloud. After that, the face’s label is determined...
Table 4. Comparison between our semi-supervised model and other model [21] on ShapeNetPart segmentation task. Average mIoU over instances (Ins.) and categories (Cat.) are reported. MGM-AE-5 stands for training the appended MLP with 5% of the training data. And MGM-AE-1 stands for 1%. Ear: Earphone. Gui: Guitar. Lap: Laptop. Roc: Rocket.

Figure 4. Visualization of test and validation accuracy under different training and test masking ratio on the graph. (a) plots the curve of test accuracy, validation accuracy, and validation loss (with unit $10^{-3}$) by fixing the masking ratio at testing to 0 and varying the training masking ratio from 0.1 to 0.9. (b) and (c) are the heat maps of test accuracy and validation accuracy. The lighter the color, the higher the accuracy. The highest test accuracy (89.830%) is masked in bold in (b).

by the major vote of all the sampled points’ labels.

After the processing, we follow [71] to randomly use 5% and 1% of the ShapeNetPart training data to evaluate the segment part task in a semi-supervised setting. We use the same pre-trained model to extract the face features of the sampled training data, along with validation and test samples without any finetuning. Following [21], we then train a 4-layer MLP [2048, 4096, 1024, 50] on the sampled training sets and evaluate it on all test data. The input feature to the MLP is the concatenation of face node embeddings and global graph embeddings which makes the input features have a dimension size of 2048. We train the model with Adam optimizer with a fixed learning rate of 0.002. This training process takes 30 epochs and converges very fast. Because the features are clear for the MLP to distinguish, the entire process takes about 15 minutes, including the testing after each epoch’s training.

During testing, we project the label computed on the meshes’ faces back to the provided point clouds according to the distance between the points and faces. Results shown in Table 4 suggest that our method is able to perform on par with the point cloud baselines and on ShapeNetPart semi-supervised learning segmentation task. In Figure 3, we show the visualization result of our semi-supervised learning segmentation. More segmentation visualization results are shown in Figure 4 in the appendix. In essence, our approach demonstrates the effectiveness of using a pre-trained model to extract face features, which are then used to train an MLP for segmentation tasks. This method converges quickly and performs on par with existing point cloud baselines.

5. Parameter Analysis

Analysis on Masking Ratio. In [22], the researchers proposed a masked autoencoder model for transfer learning tasks, where the input data is partially masked during training. The authors assumed that providing as much information as possible to the trained model during testing is the best choice and therefore, the test masking ratio was fixed at 0. In this study, we investigate the impact of test masking ratios on transfer learning tasks using a masked autoencoder model. Unlike traditional approaches where the test masking ratio is fixed, we treated it as a variable during the evaluation of the pre-trained model. Through our experiments, we discovered that the performance on the transfer learning task is affected by the training masking ratio as seen in Figure 4 (a) where we fixed the test masking ratio to 0.0 and varied the training masking ratio from 0.1 to 0.9. Furthermore, by varying the masking ratio during both training and testing in Figure 4 (b) and (c), we found that the maximum test accuracy was achieved when the training masking ratio was 0.6 and the test masking ratio was 0.1 or 0.3. This result suggests that a test masking ratio of 0 is not mandatory when evaluating a model trained with masking autoencoding and that the optimal test masking ratio is dependent on the specific downstream task and the chosen training masking ratio.
For the convenience of delivery, we denote a 2D coordinate \((a, b)\) as the situation when the training masking ratio is \(a\), and the test masking ratio is \(b\). In Figure 5, we investigate why the best test accuracy happens at \((0.6, 0.1)\) and \((0.6, 0.3)\). We compute the difference between validation accuracy and test accuracy. This difference is usually taken as the symbol of overfitting or underfitting. It turns out that in most cases, our model overfitted the task which means that validation accuracy is larger than test accuracy. But those maximum test accuracy points happen to be the points that are less overfitting. Another point that exhibits such property is \((0.7, 0.7)\) in the difference map. But at that point, more information about the mesh is lost. Three regions on the heat map in Figure 5 exhibit the less overfitting property. The last one is at \((0.2, 0.6)\). But the testing ratio is too high that the model is not overfitting but also extracts less useful information. Even though in MaskMAE, 0.75 is the best choice for masking, our 3D mesh dataset differs from the image dataset. In 3D space, a lower training masking ratio proves optimal, indicating that a mesh face in classification is more significant than individual image pixels.

In this study, we found that the point at position \((0.5, 0.5)\) in our experiments resulted in the most overfitting of the model. There are two potential explanations for this. First, training with a masking ratio of 0.5 results in a model with the highest capacity, making validation easier but testing harder. Second, having the same masking ratio for both training and testing may cause the model to rely too heavily on finding information from the masking itself, rather than the underlying features relevant to the classification task. On the other hand, the point at \((0.6, 0.1)\) had a more balanced performance. The model was trained at a masking ratio of 0.6, but tested at a masking ratio of 0.1. This helps to remove redundant information unrelated to the classification task, while also forcing the model to discard information on masking and focus on the common details relevant to the task. Additional accuracy curves under different training and test masking ratios (ranging from 0.0 to 1.0 in increments 0.1) can be found in Section 3 of the appendix.

6. Conclusion

We propose a mesh-based self-supervised learning framework that can be pre-trained on large-scale 3D imaging datasets to learn face node and shape graph features on meshes using graph masked autoencoding. We thoroughly evaluated our model on mesh classification and segmentation benchmarks. The results suggest that the learned node and graph features outperform prior state-of-the-art models. For instance, in ModelNet40 transfer learning classification tasks, our model achieved a state-of-the-art (among self-supervised mesh encoders) accuracy of 90.8% and 93.2%. We also find that different combinations of test and training masking ratios in MGM-AE could provide varying information to downstream tasks. In the ShapeNetPart segmentation task, it achieved a mIoU of 78.5, which outperforms the state-of-the-art encoders.

Our work’s novelty lies in the unique approach of leveraging the inherent topology information of mesh data. Mesh data, unlike other types of data, capture the spatial relationships and geometric properties of objects in a more detailed and structured manner. This inherent topology information provides a rich source of features that can be exploited for learning tasks. Our mesh-based masked encoding pre-training framework is designed to capture and utilize this information effectively, leading to improved performance in downstream tasks.

We believe our work opens up a new direction for mesh deep learning analysis and self-supervised learning on mesh data by demonstrating the potential of mesh data and the effectiveness of our approach.
References


[22] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.


