Walk and Learn: Self-Supervised Learning on Triangular Meshes

Gal Yefet[0009−0006−7350−1625] and Ayellet Tal[0000−0002−4967−7309]

Technion, Haifa, Israel
galyefet1919@gmail.com, ayellet@ee.technion.ac.il

Abstract. This paper addresses the challenge of self-supervised learning for 3D mesh analysis. It presents a new approach that uses random walks as a form of data augmentation to generate diverse representations of mesh surfaces. Furthermore, it employs a combination of contrastive and clustering losses. The contrastive learning framework maximizes similarity between augmented instances of the same mesh while minimizing similarity between different meshes. We integrate this with a clustering loss, enhancing class distinction across training epochs and mitigating training variance. Our model's effectiveness is evaluated using mean Average Precision (mAP) scores and a supervised linear SVM classifier on the extracted features, demonstrating its potential for various downstream tasks such as object classification and shape retrieval.

Keywords: Mesh Analysis · Random Walks · Self-Supervised.

1 Introduction

Shape analysis of three-dimensional (3D) objects plays a crucial role in contemporary research within the fields of computer vision and computer graphics. Its significance stems from its application across various domains, such as self-driving cars, virtual and augmented reality, robotics, and medicine, among others. Numerous representations exist for 3D objects, with triangular meshes, point clouds, and volumetric data being the most prominent.

This paper specifically concentrates on triangular meshes, as they are widely used in computer graphics due to their efficiency and high quality. However, 3D meshes possess unordered and irregular characteristics, which pose challenges for deep learning algorithms. Consequently, efforts have been made to rearrange the data and redefine the convolution and pooling operations in order to enable the utilization of *convolutional neural networks (CNNs)* [3, 15, 19, 22, 30].

We take inspiration from a recent framework called Meshwalker [17], which takes a different approach. Instead of utilizing convolutional neural networks, Meshwalker suggests capturing the geometry and topology of a mesh by employing random walks along its surface. Each walk is processed by a *Recurrent Neural Network (RNN)* [7, 10, 14], which gathers surface information throughout the walk. Multiple random walks, generated independently, represent a given

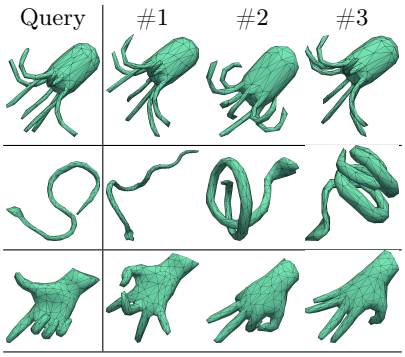


Fig. 1: **Model Retrieval Results.** This figure demonstrates the retrieval performance of our model on three query examples from SHREC11 [24]. The results show the model's ability to successfully retrieve similar items in the same class, even when the objects are presented in various poses or orientations.

mesh. Meshwalker achieves impressive results in supervised learning, demonstrating the effectiveness of randomness.

This paper, however, focuses on self-supervised learning. This is particularly relevant due to the difficulty of manual labeling of 3D data. Self-supervised learning aims to uncover patterns and extract meaningful features from data without relying on explicit labels. These extracted features can be leveraged for various purposes such as clustering, dimensionality reduction, or as a pre-training step for subsequent tasks. In the realm of two-dimensional (2D) data, self-supervised learning has been extensively explored [5, 6, 12, 13, 20, 23, 28]. In the context of three-dimensional (3D) data, several works have focused on point clouds [1, 18, 21, 26, 32]. The underlying idea involves creating augmented versions of 3D objects by applying transformations such as rotation, scaling, noise addition, etc. These augmented versions, known to be related to the same object, are then used to train the network. Some approaches employ contrastive loss [16, 18, 29], while others utilize an auto-encoder framework [1, 21, 26, 32].

Our focus, however, is on triangular meshes. Meshes are able to capture detailed local surface properties, due to the explicit neighborhoods and connectivity. While the absence of point connectivity may not pose a problem in certain conditions, there are cases in which it becomes exceedingly difficult to deduce any meaningful information from the point cloud object, as highlighted in [11].

Our key idea is to address the challenges of self-supervised learning in the context of meshes by utilizing random walks as augmentations. This may serve as a powerful mechanism to effectively represent the same model with different features. By performing random walks on both the same mesh and other meshes, we employ contrastive loss to encourage the network to learn meaningful representations for each class. This approach leverages the descriptive nature of random

walks as augmentations, enabling the network to capture and characterize the intricacies of the same model more comprehensively.

For the contrastive loss, we employ the Normalized Temperature-scaled Cross Entropy Loss (NT-Xnet) [6]. NT-Xnet loss aims to maximize the similarity between augmented instances produced from the same mesh while simultaneously minimizing the similarity between instances derived from different meshes. This prompts the network to discern fine-grained differences between classes. The random walks provide a mechanism to capture the local geometric context, while the contrastive loss leverages the relationships between the random walks, to guide the learning process. This synergy allows our solution to uncover meaningful patterns and features that can be harnessed for various downstream tasks, such as object classification, segmentation, or shape retrieval.

An additional crucial component in our architecture involves the integration of a clustering loss, which serves to enhance the notion of classes across different epochs. This approach aims to mitigate training variance by seeking proximity to the consensus established in previous epochs, rather than solely focusing on fitting to the agreement of the current epoch. The inclusion of clustering loss has been explored and showed great potential in the 2D field [5, 31].

Our model was evaluated using two self-supervised shape analysis applications: retrieval and classification. In the latter, we use a linear SVM classifier applied to the extracted features. We trained and tested our model using two commonly-used datasets: SHREC11 [24] and ModelNet40 [27]. These datasets offer diverse examples to evaluate our model's performance in learning discriminative representations for 3D shape tasks.

Figure 1 illustrates the effectiveness of the model's retrieval capabilities using three sample queries from the SHREC11 dataset: an octopus, a snake, and a hand. The displayed results highlight the model's proficiency in identifying and retrieving objects of the same class as the query, demonstrating its robustness in handling diverse poses and orientations within each object category.

This work makes three contributions:

1. The introduction of a novel self-supervised learning framework for 3D meshes.
2. The utilization of random walks as an augmentation for feature extraction in a self-supervised settings.
3. Achieving classification accuracy on the SHREC11 dataset that is only 2% lower than a supervised model demonstrates the effectiveness of our self-supervised approach in 3D shape analysis.

These contributions highlight the potential of self-supervised learning techniques in advancing the understanding and analysis of 3D meshes.

2 Model

We are provided with a collection of mesh objects in a dataset. Our model's task is to identify and extract features that are distinctive to the unknown classes from which these objects originate.

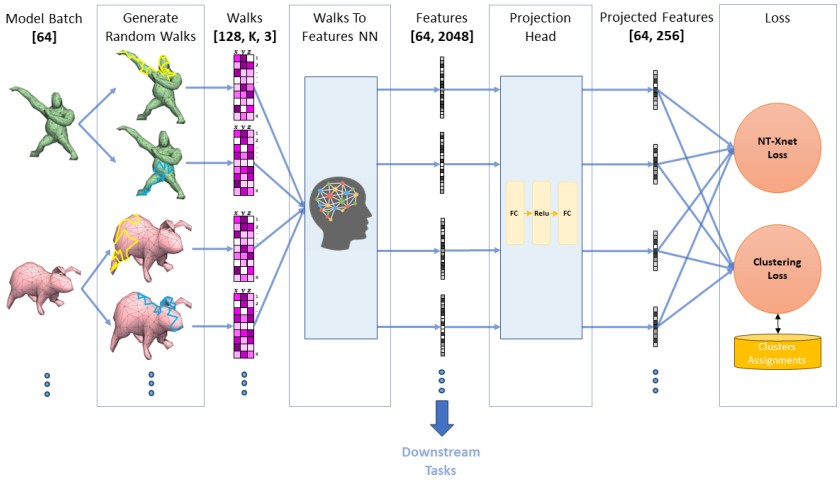


Fig. 2: **Architecture.** The model has four components: walk generation from 3D meshes, feature aggregation along each walk, a projection head for dimensionality reduction, and a combined NT-Xent and clustering loss. Features for downstream tasks are taken before the projection head.

Our model is based on the idea that embeddings of walks from the same class should be close, while those from different classes should be far apart. The problem, however, is that we do not know which objects belong to the same class. To address this, we pull together walks from the same mesh and push apart walks from different meshes. However, this does not guarantee that different objects from the same class will be clustered together. To further encourage clustering across objects of the same class, we introduce a clustering loss that promotes grouped embeddings.

Figure 2 illustrates our proposed architecture. At first, we create two random walks for every mesh. A walk is a sequence of k vertices (steps), each of which is represented by its X, Y, Z coordinates. The collected walks are fed into a *Walks-to-Features* network, which maps them into a feature space used for evaluation and downstream tasks. During training, theses features are processed through an MLP non-linear projection head. This step reduces the features to a lower-dimensional space, which is crucial for training [6]. Our losses are then computed on these features in the batch. The two loss functions we apply are the *NT-Xnet contrastive loss* and a *clustering loss,* both of which enable the model's learning of the most significant attributes for each class.

Our model consists of two parts. The first part, aiming to generate walks and embed them in space (Section 2.1). The second part learns to project features, adjusting the proximity of walks based on our key idea (Section 2.2). In the following, we will elaborate on these components. We are provided with a dataset containing a diverse set of mesh objects, with each object belonging to an unknown class.

2.1 Creating walk embedding

Data preparation. Our goal is to simplify models and create augmentations that remain within the same class. We apply classical approaches to downscale or upscale each model to augmentations with $1K$, $2K$, and $4K$ faces. This process resembles resizing an image, contributing to utilizing a uniform walk length across all models. Importantly, as we generate these variations, we retain the source model ID for each version. This assists subsequent stages in establishing connections between two distinct models.

Random walk generation. Given a batch of meshes (64 in our implementation), our goal is to create a batch of walks from which we will be able to extract qualitative features, to differentiate between various classes. Toward this end, we propose to create two random walks for each model, both sharing the same *model ID*, so that we know for certain that their features should be close, even though the walks can be very different.

Similarly to [17], each walk starts at a random vertex and proceeds by randomly selecting unvisited neighboring vertices. If all neighbors are already visited, it backtracks until it finds an unvisited one. If backtracking reaches a dead end, it jumps to a new random unvisited vertex. This captures both local and global structure of the mesh.

Walk-to-Features neural network. We are given a batch of walks, and aim at extracting qualitative features for each walk. We follow [17] and employ a *Recurrent Neural Network (RNN)* to "remember" and accumulate knowledge throughout the walk. The first sub-network consists of Fully Connected (FC) layers to convert the 3D input feature space into 256-dimensional feature space. The second sub-network is the core part, consisting of RNN layers. These three layers are fed a sequence of walk steps and output an accumulated 2048 features vector. Finally, the last sub-network involves a single FC layer to utilize the RNN's output features in creating the final features vector.

2.2 Features and Losses

At this stage, we encourage features from meshes of the same class to converge, while separating those from different classes. Recall, however, that the class memberships of the meshes are unknown. To accomplish this, we employed a contrastive loss and a new clustering loss, which complement each other as we will see below. They are applied after a non-linear projection head, which enhances the contrastive performance.

Projection Head. The goal of employing a projection head before implementing a contrastive loss is to create a denser feature space, which will be used for the contrastive loss. Previous findings, as indicated by [6], have demonstrated increased effectiveness of the contrastive loss on denser features, even when the target features precede the projection head. Applying the projection head removes information that might be useful for downstream tasks but is less crucial for the contrastive loss objective, such as the orientation of the model. In particular, we are given a batch of 64 feature vectors consisting of 2048 elements each

and our target is to compress them into 256 elements. To achieve this, we utilized a simple non-linear projection head by applying two fully connected layers with a ReLU activation function.

Loss Functions. Our aim is to apply loss functions whose gradients will drive the network to generate similar features for meshes within the same class while ensuring differences between meshes from different classes. Our primary approach involves leveraging two loss functions at different stages of the training procedure. The first one is a contrastive loss, which is very common for an unsupervised setup. The second one is a clustering loss, in which the clusters' centers are updated only every few epochs. This encourages the model to prioritize exploitation over exploration for short periods.

The contrastive loss moves each walk's features to get closer to the other augmentations of the same mesh but moves away from all other meshes. Specifically, we employ a contrastive loss known as *NT-Xnet,* introduced in SimCLR [6], and a novel clustering loss derived from the K-means algorithm. The contrastive loss aims to bring closer the features of two walks from the same mesh while moving them away from all other walks' features in the batch.

The NT_Xnet loss is defined as follows:

$$NT\_XentLoss = -\frac{1}{2N}\sum_{i=1}^{N}\log\left(\frac{\exp(\text{sim}(x_i, x_i')/\tau)}{\sum_{j=1}^{2N}\exp(\text{sim}(x_i, x_j)/\tau)}\right) \tag{1}$$

where N is the number of walks in a batch, x_i and x_i' are embeddings of a positive pair, $\text{sim}(a, b)$ is the cosine similarity between vectors a and b, and τ is the temperature parameter.

While this loss may be effective in certain scenarios, it lacks the ability to retain knowledge between batches. The clustering loss, which is discussed hereafter, addresses and resolves this issue.

We propose to maintain clusters that are updated only from time to time so that they push the model to exploit a given configuration, rather than exploring a new one every batch. Each cluster should represent the set of features associated with a single class. We utilized the K-means algorithm for updating the cluster centers and the objective function. In our case, K is the number of clusters, which is not necessarily the number of classes, as this is unknown in an self-supervised setting. The loss function of our K-means loss aims to minimize the within-cluster sum of squares (WCSS):

$$KMeans\_Loss = \sum_{i=1}^{k}\sum_{j=1}^{n_i}\|x_j^{(i)} - \mu_i\|^2, \tag{2}$$

where $x^{(i)}$ are all the feature vectors assigned to the mean μ_i.

There are three stages when using the clustering loss, as follows.

1. Initialization: We initialize our clusters by placing the first cluster randomly and then choosing subsequent clusters with a higher probability of being far away from existing clusters.

2. Assignment: Given the i-th positive pair of feature vectors, (x_1^i, x_2^i), which are the features of two walks from the same mesh, we assign them both to a single mean, which is the closest to one of them. In particular,

$$\mu^i = argmin_{\mu \in U}(min(||x_1^i - \mu||^2, ||x_2^i - \mu||^2)), \tag{3}$$

where U is the set of all means and μ^i is the chosen cluster for the i-th positive pair. Ideally, there exists a single relevant cluster center for a particular class, making the closest mean a reasonable choice.

3. Means update, as follows: From time to time, the means are updated by taking the mean of all feature vectors assigned to each cluster:

$$\mu_i = \frac{1}{n_i} \sum_{j=1}^{n_i} x_j^{(i)}, \tag{4}$$

where μ_i is the mean to update and n_i is the number of feature vectors assigned to this mean.

In practice, we initialize this loss with 80 means and updated these means after every 5 epochs. We use the clustering loss after 50 epochs, as we primarily use this function to aid in exploitation, considering that most exploration has already been facilitated by the contrastive loss.

We chose to use a $1:1$ ratio between the weight of the clustering loss and the contrastive loss when both are applied after 50 epochs, as follow ($\alpha = 1.$):

$$Loss = NT\_XentLoss + \alpha * KMeans\_Loss \tag{5}$$

2.3 Inference

At inference, our goal is to generate a feature vector for each given mesh. To achieve this, multiple walks are employed for the given mesh, with each walk producing a vector of features that represents the mesh. We assume that some walks may not capture unique aspects of the mesh, so we aim to eliminate these less informative walk features. Therefore, only half of the walks, those closest to the average of all walk feature vectors, are selected. These chosen walks are then averaged to produce the final feature vector representing the model.

To grasp the significance of walk selection and averaging, let's consider walks on a camel. Since walks are generated randomly, we anticipate that some will explore typical parts of the model, such as the body, which are similar to horse body. On the other hand, other walks are likely to explore unique parts, such as the hump or the head. The selective choosing and averaging process will most likely capture the features of a camel.

3 Results

We evaluate our model on two downstream tasks: model retrieval and clustering (classification). In our implementation, meshes are remeshed to 500 faces using Open3D quadric decimation, and the network is built in TensorFlow 2.x and trained with NT-Xent loss using random walks of length 100, 200, and 400.

Method	Split-16	Split-10
Ours (self-supervised)	94.1%	89.8%
MeshWalker (supervised)	96.8%	94.3%
GWCNN (supervised)	96.0%	87.0%

Table 1: **Results (mAP) on SHREC11.** Our method achieves good results in both splits, comparable to supervised models.

Method	mAP score
Ours (self-supervised)	71.3%
AttWalk [4] (supervised)	91.2%
MeshWalker [17] (supervised)	87.7%
MeshNet [9] (supervised)	81.9%
GWCNN [8] (supervised)	59.0%

Table 2: **Results on ModelNet40.** Supervised models outperform our self-supervised model due to high intra-class variability.

3.1 Model Retrieval

Given a query mesh, the goal is to accurately identify and retrieve similar meshes. For a given mesh, we generate multiple random walks, which are processed through the trained network. The network extracts a feature vector for each walk, and these vectors are then averaged into a single feature vector. Objects in the dataset are considered similar if their Euclidean distances to the given model are small. In practice, we use 32 walks per mesh model.

Datasets. We applied our model to two datasets: SHREC11 [24] and ModelNet40 [27]. SHREC11 consists of 30 classes, with 20 examples per class. Modelnet40 contains 12,311 CAD objects from 40 categories, out of which 9,843 objects are used for training and 2,468 are used for testing. Unlike the SHREC11 dataset, many objects in this collection consist of multiple components and are not necessarily watertight, posing challenges for some mesh-based methods.

Evaluation metric. The mean Average Precision (mAP) calculates the average precision (AP) for each query and then averages these values across all test queries. mAP provides a single-figure measure of quality across recall levels, making it a popular choice for evaluating retrieval systems.

Results. Following [11, 4, 17], we split each SHREC11 class into 16 (/10) training and 4 (/10) testing examples. Table 1 presents our performance. As the first self-supervised approach on this dataset, we compare to supervised methods, namely MeshWalker [17] and GWCNN [8]. For both splits (16/4 and 10/10), our model achieves competitive performance, with only a 2.7% gap from supervised methods. Table 2 shows the same experiment on ModelNet40. As no self-supervised results exist for this dataset, we compare to supervised models. In this case, our performance falls below that of the supervised approaches. We attribute this gap to the difficulty self-supervised models face on small datasets with high variability among objects within the same class.

Figures 1,3 present qualitative examples of objects retrieved by our model for queries from SHREC11. The retrieved objects belong to the same class as the query, even when they were in different poses. This indicates that our model has learned to recognize the general features of the classes, rather than simply memorizing specific instances.

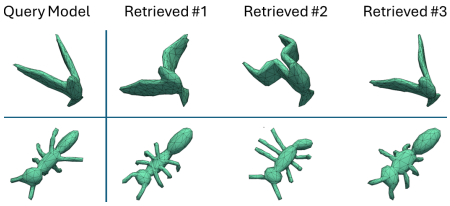
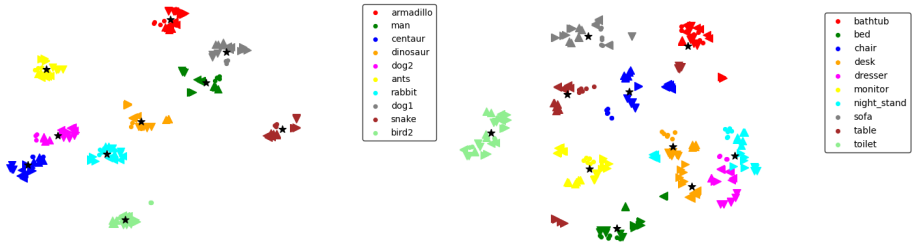


Fig. 3: **Object Retrieval.** Given a query (on the left), our model retrieves objects from the same class as the query object, even when the pose (for the bird) or head orientation (for the ant) varies.



(a) **SHREC11.** (b) **ModelNet40.**

Fig. 4: **t-SNE of model features.** Visualizations for SHREC11 and ModelNet40 (10 classes each). The plots demonstrate that objects from the same class are grouped together. Notably, clusters representing similar classes, such as different dog breeds, are positioned close to each other in the feature space.

3.2 SVM Classifier

Another common evaluation for self-supervised methods is training a linear SVM on the learned features [1, 2, 26]. The SVM finds the hyperplane that maximally separates classes in the feature space. During training, the SVM learns the weights of the linear decision boundary. Once trained, the SVM can efficiently classify new, unseen data points.

Each mesh is processed by our trained network, and features are obtained by averaging multiple walks. The resulting vectors are fed into an SVM classifier. The dataset is split into training and test sets.

We evaluate the SVM by classifying meshes from SHREC11 and ModelNet40. Performance is measured by accuracy, i.e., the proportion of correct predictions. To ensure robustness, we perform multiple runs and average the results.

Figure 4 shows a 2D t-SNE of 10 classes per dataset with cluster centers. The clear separation suggests a linear classifier can distinguish classes. Each class clusters around a black star, with similar classes (e.g., 'dog1' and 'dog2') appearing close in feature space.

While most classes form distinct clusters, a few exhibit noise and overlap. In particular, the 'desk,' 'dresser,' and 'night-stand' classes show some confu-

Method	Accu.
Ours (self-supervised)	95.5%
MeshWalker (supervised)	97.5%

Table 3: **SVM classification on SHREC11.** Features extracted by our method yield competitive results using a linear SVM.

Method	Input	Accu.
Ours (self-supervised)	Mesh	79.5%
MeshWalker (supervised)	Mesh	91.3%
Crosspoint (self-supervised)	Points	91.2%
[1] (self-supervised)	Points	84.5%
[26] (self-supervised)	Points	83.3%

Table 4: **SVM classification on ModelNet40.** Comparison to SOTA point-cloud methods, as no mesh-based works exist.

sion. Although this reflects imperfect linear SVM performance, the confusion is expected due to strong feature similarities. For this dataset, our self-supervised model struggles more than a supervised approach to determine whether two samples from these similar classes belong to the same class. This difficulty may be attributed to class imbalances, which pose a challenge for contrastive models [25].

Table 3 reports SVM classification accuracy on SHREC11 using features from our unsupervised network and the supervised MeshWalker. In both cases, these features were then used to train the linear SVM classifier, which we applied to the test data. It shows a competitive results with a fully supervised model, with only a 2% difference between our self-supervised model and the supervised MeshWalker classification accuracy using a linear SVM.

Table 4 reports our ModelNet40 classification accuracy compared with the supervised MeshWalker method. Our framework also enables comparison with existing self-supervised approaches on ModelNet40 point clouds, which we include since such methods are available for point-cloud data. Our model's performance on ModelNet40 is 11.7% lower than CrossPoint [2]. We believe this gap is due to our use of a contrastive approach in the loss function and challenges with unbalanced datasets. The issue arises because the number of 'False Negatives' in each batch varies greatly depending on the class.

4 Conclusion

This paper has introduced a self-supervised learning system for 3D triangular meshes, addressing a gap in the field of 3D shape analysis. The key idea is to utilize random walks as a novel method of augmentation for 3D mesh models, enabling the extraction of high-quality features without the need for labelled data. This approach effectively tackles the challenges posed by the unordered and irregular nature of 3D meshes in deep learning algorithms.

Utilizing these random walks, we proposed an end-to-end learning framework that leverages contrastive learning techniques and clustering loss. The random walks serve as a powerful mechanism to capture local geometric context, while the contrastive and clustering loss guides the learning process by maximizing

similarity between augmented instances of the same mesh and minimizing similarity between instances from different meshes. This synergy allows our solution to uncover meaningful patterns and features in 3D mesh data.

Finally, this approach proves effective for downstream tasks, with features supporting two key applications in 3D shape analysis: object classification and shape retrieval. Notably, our approach achieved excellent clustering results when evaluated on SHREC11, a small and balanced dataset, showing that the proposed method is promising. However, the results on ModelNet40 were less favorable, likely due to the nature of its classes and the imbalance between them.

While our results show strong feature learning, future work should address limitations on ModelNet40. Ablation studies could validate our design choices, and replacing the RNN with an attention-based transformer is another promising direction. Exploring additional mesh-based tasks, especially with limited labels, is also worthwhile.

Acknowledgments: This work was supported by the Israel Science Foundation (ISF) under grant 2329/22.

References

1. Achlioptas, P., Diamanti, O., Mitliagkas, I., Guibas, L.: Learning representations and generative models for 3D point clouds. In: ICML. pp. 40–49 (2018)
2. Afham, M., Dissanayake, I., Dissanayake, D., Dharmasiri, A., Thilakarathna, K., Rodrigo, R.: Crosspoint: Self-supervised cross-modal contrastive learning for 3D point cloud understanding (2022), https://arxiv.org/abs/2203.00680
3. Atzmon, M., Maron, H., Lipman, Y.: Point convolutional neural networks by extension operators. ACM Transactions on Graphics (TOG) **37**(4), 1–12 (2018)
4. Ben Izhak, R., Lahav, A., Tal, A.: AttWalk: attentive cross-walks for deep mesh analysis. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV). pp. 299–308 (2022)
5. Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., Joulin, A.: Unsupervised learning of visual features by contrasting cluster assignments. Advances in Neural Information Processing Systems **33**, 9912–9924 (2020)
6. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: ICML. pp. 1597–1607 (2020)
7. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation (2014)
8. Ezuz, D., Solomon, J., Kim, V.G., Ben-Chen, M.: Gwcnn: A metric alignment layer for deep shape analysis. Computer Graphics Forum **36**(5), 49–57 (2017)
9. Feng, Y., Feng, Y., You, H., Zhao, X., Gao, Y.: Meshnet: Mesh neural network for 3D shape representation (2018), https://arxiv.org/abs/1811.11424
10. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. **31**(5), 855–868 (2009)
11. Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., Cohen-Or, D.: MeshCNN. ACM Transactions on Graphics **38**(4), 1–12 (2019)
12. Henaff, O.: Data-efficient image recognition with contrastive predictive coding. In: International conference on machine learning. pp. 4182–4192. PMLR (2020)

13. Hjelm, R.D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., Bengio, Y.: Learning deep representations by mutual information estimation and maximization. arXiv preprint arXiv:1808.06670 (2018)
14. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
15. Hua, B.S., Tran, M.K., Yeung, S.K.: Pointwise convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 984–993 (2018)
16. Jiang, J., Lu, X., Ouyang, W., Wang, M.: Unsupervised representation learning for 3D point cloud data. arXiv preprint arXiv:2110.06632 (2021)
17. Lahav, A., Tal, A.: MeshWalker: deep mesh understanding by random walks. ACM Transactions on Graphics (TOG) **39**(6), 1–13 (2020)
18. Li, X., Yu, L., Fu, C.W., Cohen-Or, D., Heng, P.A.: Unsupervised detection of distinctive regions on 3D shapes. ACM Transactions on Graphics (TOG) **39**(5), 1–14 (2020)
19. Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B.: Pointcnn: Convolution on $\mathcal{X}$-transformed points (2018)
20. Oord, A.v.d., Li, Y., Vinyals, O.: Representation learning with contrastive predictive coding. arXiv preprint arXiv:1807.03748 (2018)
21. Sharma, A., Grau, O., Fritz, M.: Vconv-dae: Deep volumetric shape learning without object labels. In: European conference on computer vision. pp. 236–250. Springer (2016)
22. Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J.: Kpconv: Flexible and deformable convolution for point clouds (2019)
23. Tian, Y., Krishnan, D., Isola, P.: Contrastive multiview coding. In: European conference on computer vision. pp. 776–794. Springer (2020)
24. Veltkamp, R.C., van Jole, S., Drira, H., Amor, B.B., Daoudi, M., Li, H., Chen, L., Claes, P., Smeets, D., Hermans, J., et al.: SHREC'11 track: 3D face models retrieval. In: 3DOR. pp. 89–95 (2011)
25. Vito, V., Stefanus, L.Y.: An asymmetric contrastive loss for handling imbalanced datasets. Entropy **24**(9), 1303 (Sep 2022)
26. Wu, J., Zhang, C., Xue, T., Freeman, B., Tenenbaum, J.: Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. Advances in neural information processing systems **29** (2016)
27. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3D Shapenets: a deep representation for volumetric shapes. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1912–1920 (2015)
28. Wu, Z., Xiong, Y., Yu, S.X., Lin, D.: Unsupervised feature learning via non-parametric instance discrimination. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3733–3742 (2018)
29. Xie, S., Gu, J., Guo, D., Qi, C.R., Guibas, L., Litany, O.: Pointcontrast: Unsupervised pre-training for 3D point cloud understanding. In: European conference on computer vision. pp. 574–591. Springer (2020)
30. Xu, Y., Fan, T., Xu, M., Zeng, L., Qiao, Y.: Spidercnn: Deep learning on point sets with parameterized convolutional filters (2018)
31. Yan, X., Misra, I., Gupta, A., Ghadiyaram, D., Mahajan, D.: Clusterfit: Improving generalization of visual representations (2019)
32. Yang, Y., Feng, C., Shen, Y., Tian, D.: Foldingnet: Point cloud auto-encoder via deep grid deformation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 206–215 (2018)