AttWalk: Attentive Cross-Walks for Deep Mesh Analysis

Ran Ben Izhak
Technion, Israel
benizhakran@gmail.com

Alon Lahav
Technion, Israel
alon.lahav2@gmail.com

Ayellet Tal
Technion, Israel
ayellet@ee.technion.ac.il

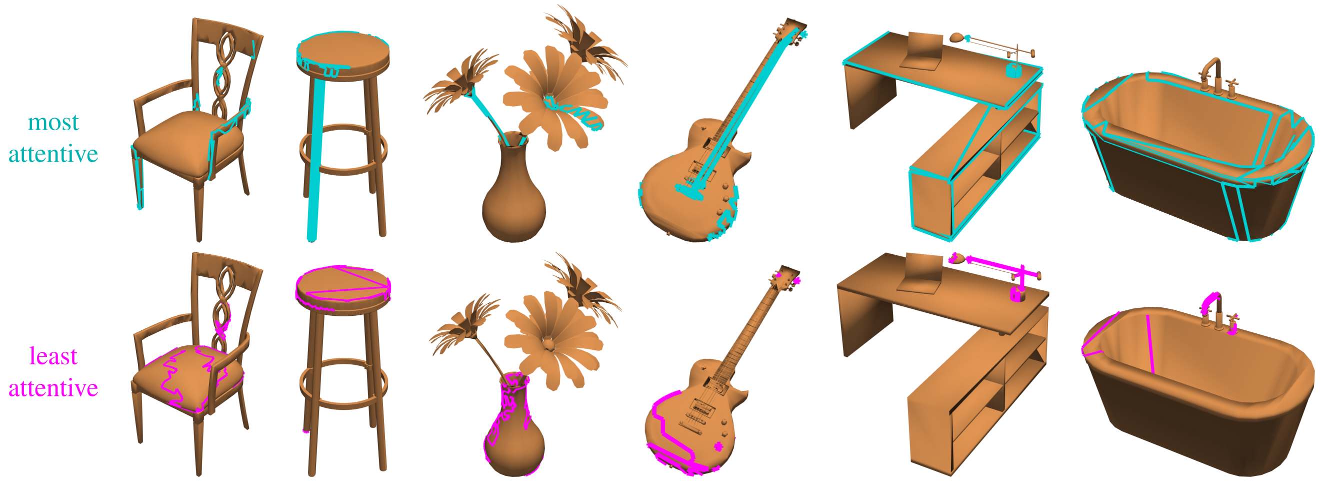

Figure 1: **Most & least attentive walks.** A set of random walks over a surface is a successful representation of meshes for deep learning. Which walks contribute more to the representation? The most attentive walks (in cyan) provide a general "view" of the object and explore its distinctive features, e.g. the guitar's neck and strings. In contrast, the least attentive walks (in magenta) focus on regions that do not distinguish the object from others, e.g. the round seat of the stool.

Abstract

Mesh representation by random walks has been shown to benefit deep learning. Randomness is indeed a powerful concept. However, it comes with a price—some walks might wander around non-characteristic regions of the mesh, which might be harmful to shape analysis, especially when only a few walks are utilized. We propose a novel walk-attention mechanism that leverages the fact that multiple walks are used for a single mesh representation. The key idea is that the walks may provide each other with information regarding the meaningful (attentive) features of the mesh. We utilize this mutual information to extract a single descriptor of the mesh. This differs from common attention mechanisms that use attention to improve the representation of each individual descriptor. Our approach achieves SOTA results for two basic 3D shape analysis tasks: classification and retrieval. Even a handful of walks along a mesh suffice for learning. Furthermore, our approach provides insight into mesh importance detection.

1. Introduction

Shape analysis of 3D objects is a fundamental aspect in modern computer vision and computer graphics research. This is due to the paramount importance of shape analysis to numerous applications, including self-driving cars, virtual & augmented reality, robotics, medicine and many more.

There are several representations of 3D objects, most notably triangular meshes, point clouds and volumetric data. This work focuses on triangular meshes, which are the most common representation in computer graphics, thanks to its efficiency and high-quality. Unfortunately, 3D meshes are unordered and irregular, which is challenging for deep learning algorithms. This has led to attempts to "re–order" the data and re-define the convolution & pooling operations, in order to be able to utilize CNNs [14, 22, 53, 55].

Instead of re-ordering the unordered data to suit CNNs, a different approach was recently proposed [31]: The idea is to capture the geometry & topology of a mesh, by simply walking along its surface in a random manner. The properties of the walk can be aggregated by a *Recurrent Neural Network (RNN)*. A given mesh can then be represented by several independent random walks.

Inspired by this approach, we show how to overcome its major drawback by utilizing its unique characteristic. In particular, while randomness is powerful, it might produce walks that do not represent the mesh well, sometimes leading to failures even when many walks are used. However, the fact that a mesh is represented by multiple walks helps us to focus on the important properties of the walk—those that distinguish the mesh from others.

We propose to learn how to weigh the features of the various walks, exploiting the fact that they all represent the very same mesh. These walks may provide each other information on the important features of the walk and jointly derive a good mesh description. For intuition sake, let us draw an analogy to sentences. Suppose that we are given different sentences describing the same event (i.e., various walks describing the same mesh). Our goal is come up with a single description of the significant features of the event, by utilizing the collection of sentences.

This idea is related to the notion of attention, which is popular in NLP and in image processing, but is sparser in mesh analysis in 3D [33, 38, 61]. However, differently from other attention mechanisms, we interpret the attention vectors as cross-walk probabilities (rather than as new feature vectors). This enables us to use them, jointly with the original feature vectors, to generate a single mesh descriptor. This descriptor focuses on the distinctive mesh features encountered by specific random walks, while neglecting the effect of commonplace features encountered by others. In our event analogy, rather than improving the descriptors of each sentence separately, our attention uses the various attentions to generate a single event descriptor.

Interestingly, our novel walk-attention yields insight regarding the distinctiveness of mesh regions. Figure 1 illustrates the most attentive walks—-those that influence the final mesh descriptor the most (and similarly, the least attentive walks). The most attentive walk of the stool, as determined by our model, goes both through the seat and a leg, whereas the least attentive walk goes only through the seat, and thus misses information needed for classification. Similarly, the most attentive walk of the guitar explores the neck and the strings, and that of plant explores the flowers (neglecting the walk on the vase).

We evaluate our method for two fundamental shape analysis applications: mesh classification and mesh retrieval. We show that our model achieves SoTA results, using significantly few walks for certain datasets. For instance, for Modelnet40, $\frac{1}{8}^{th}$ of the walks suffice, compared to [31], while the results improve. This is thanks to focusing on the most distinctive portions of the walks, rather than averaging all walks. Hence, the paper makes two contributions:

1. We introduce a novel attention mechanism to deep learning on meshes. This mechanism also provides insight into the regions of 3D objects that are more (or less) important than others for shape analysis tasks.
2. We present an end-to-end learning framework that realizes this attention. It achieves SoTA results for 3D shape classification and retrieval.

2. Related work

Mesh deep learning. A triangular mesh is the most widespread 3D representation in computer graphics, used in virtual reality, CAD, medical applications etc. A mesh consists of sets of vertices $\mathcal{V}$, edges $\mathcal{E}$ and faces $\mathcal{F}$. Since each vertex has a different number of neighbors, at different distances, the basic question is how this irregular representation shall be handled within deep learning.

In an attempt to "re–order" the data, it was suggested to convert the mesh into volumetric grids [4, 13] or into multiple 2D projections (multi-view) [15, 30, 48, 54]. Point clouds have been handled quite intensively as well, resulting in interesting convolution and pooling operators [1, 42, 44, 51, 60]. Recently, implicit functions have also been proposed [17, 29, 37, 40]. See [18] for a thorough review.

To handle meshes directly, novel convolutions and/or vertex neighborhoods have been defined [14, 19, 41, 46, 53]. Other works parameterize the mesh in 2D [5, 12, 25, 36, 47]. In [22], a unique idea of using the edges of the mesh to perform pooling and convolution, is introduced.

Our work is based on a different idea, of representing a mesh by a set of random walks over its vertices, along the edges [31]. Each vertex of the walk is represented by the 3D coordinates offset from the previous vertex of the walk. The walk is fed into a *Recurrent Neural Network (RNN)* that "remembers" the walk's history.

3D attention. Attention was first introduced in [2] for language translation and has since revolutionized *natural language processing* [8]. This success has inspired the applications of attention to image analysis tasks, such as in recognition [11, 26, 66], synthesis [64], and captioning [59, 63].

In 3D, attention is used within multi-view representation, either aggregating features by attention in consecutive views [20, 23, 54] or, in addition, selecting the next views according to attention [7, 21]. Recently, attention has been used for point clouds, attempting to capture the local [27, 65] or the global [39, 50] context of a point. Others have learned contextual relation between point patches [57]. Point transformers have also been proposed [58, 62].

The work on mesh attention, however, is sparser. It was used for reconstructing 3D human pose [33] or for mesh deformation [61]. In classification and segmentation, PD-Meshnet [38] extend the primal-dual graph framework to 3D meshes, utilizing a graph attention network to capture global context. The non-local nature of transformers is exploited in [33]. Attention masks are extracted in [61] in

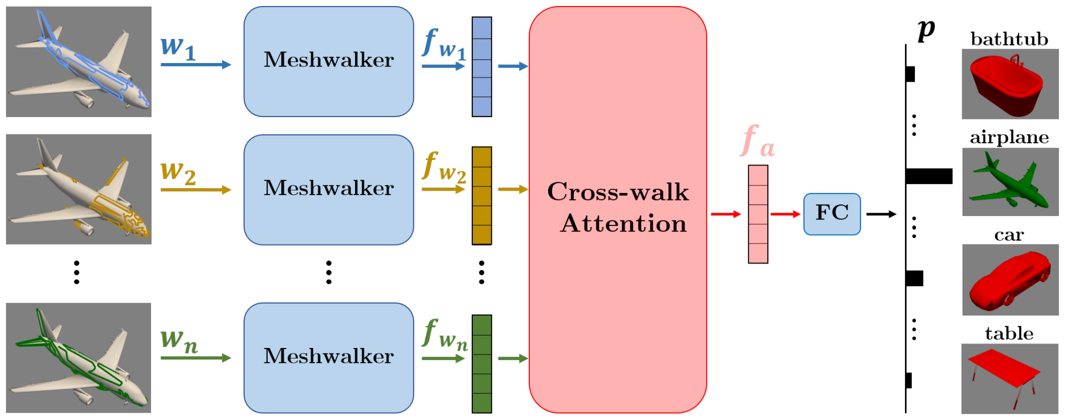

Figure 2: **Architecture.** Each walk, w_i, is processed independently by *Meshwalker* [31], excluding its last classification layer, resulting in a feature vector, f_{w_i}, for each walk. These n feature vectors are the input to our novel attention module, which produces a single mesh feature vector, f_a, which emphasizes the most attentive properties of the mesh. The last fully-connected layer transforms f_a into a probability vector, which is used for shape analysis applications (e.g. p is a prediction vector for classification). Figures 3 and 4 illustrate the architectures of MeshWalker and the cross-walk attention module.

order to attend different shape parts at lower scale, enabling fine part-deformation for local attentive regions.

Our proposed attention is inherently different. It operates on multiple random walks on a mesh, using attention to learn to produce a single cross-walk attentive features.

3. Model

We wish to learn how to separate the wheat from the chaff, focusing on the mesh relevant features and ignoring the irrelevant ones. For instance, to distinguish a chair from a stool in Figure 1, it is beneficial to explore the backrest (or the lack of it) and not the seat. Generally, in Figure 1 it is better to focus on the features along the cyan walks than on the features along the corresponding magenta walks.

We propose a novel model that benefits from having multiple pieces of information that explore the mesh in diverse manners. Our key idea is that, given multiple feature vectors, learned from their respective walks, we will learn to focus on the most informative entries of these descriptors. This is done by a novel *Many-to-One (MtO)* attention module that generates a single feature vector from the multiple sources. This descriptor highlights the informative features and neglects the others. Furthermore, our attention is invariant to the number of walks and to any possible relation between them (in contrary to multi-view attention, for example, which might require fixed order and distance between consecutive views, as well as a constant number of views).

In analogy, consider multiple sentences that describe the same event from different perspectives (multiple walks that describe the same mesh). Each sentence is processed separately and a feature vector is generated for it. Our goal is to learn a single feature vector that describes the event as a whole, by utilizing the collection of feature vectors generated for the sentences. This is different from common attention units [52], where attention is used to improve the individual descriptors. We note that this analogy is not complete, as randomness is unique to our case.

Figure 2 illustrates the architecture of our proposed model. Hereafter, we briefly describe each of its components, while elaborating on the attention module, which is the key of our framework.

Random walks. Given a mesh, n random walks, w_i, $1 \leq i \leq n$ are generated for it. Briefly, as in [31], a random walk is defined as a sequence of vertices: The first vertex is selected randomly, and then the next vertices are added iteratively, where each vertex is chosen randomly from the vertices adjacent (along an edge) to the current one. Each walk vertex is represented as the 3D translation from the previous vertex. Thus, each walk wanders around the mesh, going through its ridges and valleys, exploring its meaningful, as well as its non-meaningful parts.

MeshWalker. Given a random walk, we apply the *MeshWalker* network [31], for each walk separately. Our model consists of n instances of MeshWalker, excluding its last classification layer. Each instance processes a single walk, w_i, independently, generating a feature vector for this walk f_{w_i}, $1 \leq i \leq n$. As illustrated in Figure 3, MeshWalker first learns to map each walk vertex to a new feature space in high dimension, by fully-connected layers. Then, a *Recurrent Neural Network (RNN)*, having a hidden state vector ("memory") that contains the information gathered along the walk, is applied. It learns to accumulate the important information along the walk and forget the non-important information. The RNN is implemented using three *Gated Re-*

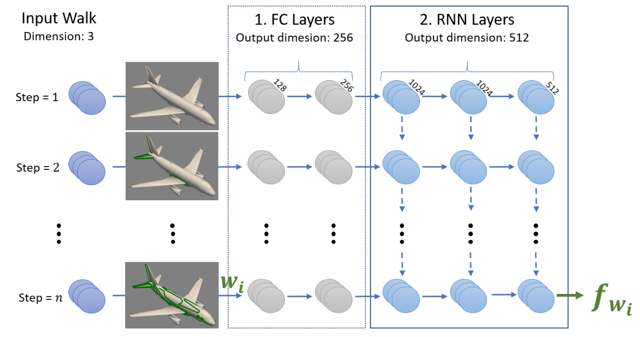

Figure 3: **Meshwalker [31].** This network gets as input a random walk (a sequence of vertices) along the mesh, w_i. Each vertex is first embedded into a higher dimension feature vector by two fully-connected layers. Then, subsequent three RNN (GRU) layers process the sequence of feature vectors into a single walk feature vector, f_{w_i}, which describes the properties of the walk.

current Units (GRU) [9]. Thus, a different feature vector is learned for each walk, describing the mesh from the specific walk's perspective.

Cross-walk attention. Our proposed cross-walk attention module is illustrated in Figure 4. It gets as input n feature vectors, f_{w_i}, and its goal is to learn how to generate a single feature vector that describes the mesh, f_a, such that the important information from all the walks are aggregated.

We may think of every vector's entry as describing a certain property of the walk. Thus, in order to compute entry i in the resulting mesh feature vector, all the i^{th} entries of the walk feature vectors should be used, independently of the other vectors' entries. This is done by learning a weight for each entry of each feature vector. For learning each weight, however, all entries from all feature vectors should used. Finally, the weight vectors, jointly with the walk feature vectors, are used to generate the final result. In the following, we will elaborate on the details of this idea. However, we note already that our scheme may be beneficial in other scenarios where the same object/scene/event can be described in diverse, though tightly related, manners.

Our attention block consists of two sub-blocks. The first sub-lock uses the attention information to improve the representation of each individual feature vector, as commonly done. The novelty of our attention module is the second sub-block, which combined the information into a single mesh attention vector.

In particular, the first sub-block is given n walk feature vectors $\{f_{w_i}\}_{i=1}^{n}$, stacked to form a matrix F_w. The goal is to generate a new matrix of walk features, H_a, in which each walk is enriched by information from other walks. Given F_w, we learn its self-attention map, using the scaled dot-product attention block of [52], which transforms each walk features into a self-attention vector. Briefly, F_w is transformed into three feature sub-spaces Q, K, V. Intuitively, the columns of Q and K are walks descriptors, learned to represent relevancy to one another. While Q and K play a similar role, their different walk descriptors enable the walks to influence one another in a non-symmetric manner. V's columns represent intra-walk attention per entry. Q, K, V are computed as follows:

$$Q = W^{(q)} F_w, \quad K = W^{(k)} F_w, \quad V = W^{(v)} F_w, \tag{1}$$

where $W^{(q)}, W^{(k)}, W^{(v)} \in \mathbb{R}^{d \times d}$ are learned weight matrices that are used to project the walk feature vectors into different sub-spaces of dimension d. (In our implementation $d = 512$, the same dimension as that of the walk feature vectors). The self-attention weights are computed as

$$W_{sa} = softmax(\frac{K^T Q}{\sqrt{d}}). \tag{2}$$

Finally, the self-attention feature vectors are aggregated in matrix H_a. Each column of H_a utilizes knowledge from all the walks and will be used next to weigh the importance of the different features of each walk, in order to represent the mesh as a whole. H_a is defined as

$$H_a = V W_{sa}. \tag{3}$$

The second sub-block (the cross-walk attention sub-block in Figure 4) aims at aggregating the walk features in F_w into a single mesh feature vector f_a. This is done first giving H_a, in which each entry indicates the importance of that entry, a probabilistic interpretation, by applying softmax per row i. These probabilities are then multiplied, element-wise, by the walk feature vectors F_w, to create the cross-walk attention features. That is to say, every row is weighted (cross-walk) according to the learned probabilities of each walk. Finally, the columns j of the cross-walk attention matrix are summed, creating the sought-after mesh feature vector, f_a. This procedure is expressed as

$$G = F_w \odot softmax(H_a) \tag{4}$$
$$(f_a)_i = \sum_{j=1}^{n} G_{i,j}.$$

This fine aggregation of multiple walks is at the core of the success of our attention. This attention is invariant to the order of the walks and to the choice of n, the number of walks per object.

Note that f_a resides in the original walk feature subspace, since each of its entries is a linear combination of the corresponding entries of F_w. This enables the classification layer in Figure 2 to process both single-walk features and cross-walk attentive features seamlessly. This is essential to our two-phase training, which is described next.

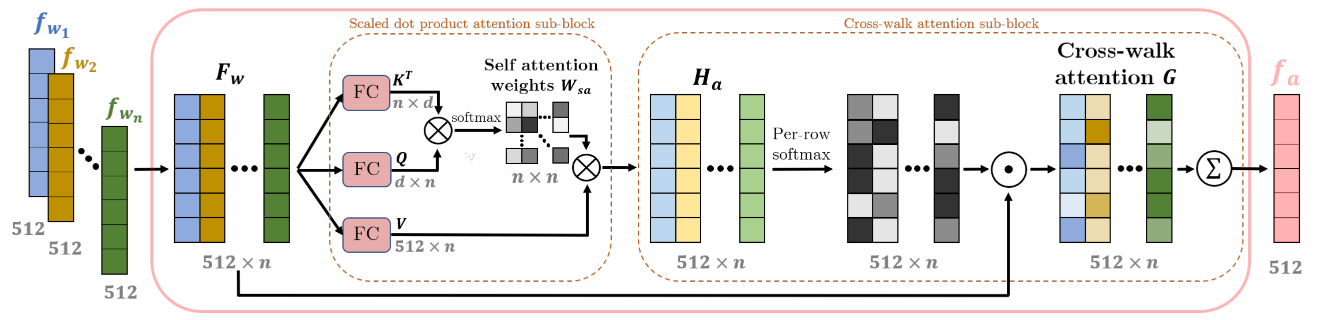

Figure 4: **Cross-walk attention.** Given n feature vectors f_{w_i}, each representing a walk w_i, we compute a cross-walk attention vector f_a. At first, the *scaled dot product attention* of [52] is applied to the input vectors. It starts by utilizing 3 parallel fully-connected layers per walk: the first two learn the attention between each walk vector to the other walk vectors, and the third transfers the input walk to be multiplied by a function of the former two. The output of this sub-block are attention feature vectors for the walks, denoted as H_a. The second sub-block generates a single vector that represents the mesh in a way that weighs the importance of each walk to each entry. This is done in three steps, given n walk attention feature vectors H_a: (1) *softmax* is applied per row in order to transform it into a weight (probability) vector. (2) Hadamard product ($\odot$) between the acquired weights and the input walk features scales each feature entry according to its learned importance. (3) The weighted walk features are summed across the walks, to produce the output feature vector f_a.

Fully-connected layer. The final prediction vector, p, is generated by a fully-connected layer, given f_a as input.

Training. We pursue a two-phase training strategy. The first phase attempts to extract the best features per walk independently. Thus, the network (Figure 2) is trained without our cross-walk attention and learns to extract meaningful features & to correctly classify the shape by every single walk. In the second phase, we freeze the MeshWalker block (the cyan rectangles in Figure 2) and train the attention block, so as to account for the most relevant features across all walks. The two-phase training prevents the network from focusing only on the most attentive walks, which would result in avoiding to process mesh regions that are less attentive, but may be important for fine results. In Section 5 we compare this strategy to 1-phase training.

Recall that both the individual walk features f_{w_i} and the mesh feature vector f_a reside in the same sub-space, thus both phases can be trained by the same loss. Given a prediction vector (either per-walk in Phase 1 or p of Figure 2 in Phase 2) and the corresponding class label l, training is performed by minimizing the Softmax cross entropy loss

$$L(p,l) = -\log \frac{e^{p_l}}{\sum_{j=1}^{C} e^{p_j}}. \tag{5}$$

For retrieval, we train our model using a combination of the softmax cross entropy loss and the triplet-center loss (*TCL*) [24]. Intuitively, *TCL* attempts to push each walk prediction vector closer to its corresponding class center and away from centers of other classes. Specifically, given p and l as above, the TCL is defined as

$$TCL(p,l) = \max\left(D(p,c_l) + m - \min_{k \neq l} D(p,c_k), 0\right). \tag{6}$$

Here, each class is represented by a learned parametric center c_k (of the same dimension as p). The margin m is a hyper parameter that prevents pushing the vector too far ($m = 1$ in our experiments). $D(\cdot)$ is the squared Euclidean distance. The combination of the losses is defined as

$$\mathcal{L}(p,l) = \lambda_1 TCL(p,l) + \lambda_2 L(p,l). \tag{7}$$

In all our experiments $\lambda_1 = 1$ and $\lambda_2 = 0.01$. We note that though this loss encourages the network to learn more discriminative mesh features and indeed improves retrieval results, empirically it does not improve classification results.

Implementation details: Training is performed in batches of $M = 64$ walks, where a batch contains walks from several meshes. In the first phase each walk belongs to a different mesh, whereas in the second phase we use 8 walks per mesh, for 8 meshes. All the meshes in our experiments are normalized and simplified into $1K$, $2K$ and $4K$ faces, both to reduce the network capacity required for training and as a form of data augmentation. At inference, we average the scores of the predictions at the different scales. Meshes with less faces than the above scales are used without simplification. In the first training phase, we use Adam optimizer with cyclic learning rate of $5 \cdot 10^{-4}$ to 10^{-6} with $20K$ iterations per cycle, for a total of $200K$ iterations. In the second phase, we reduce the learning rate by half, which is more stable for fine-tuning the cross-walk attention block, training for additional $100K$ iterations.

4. Applications

The performance of our model is evaluated on 3D shape classification and retrieval, for a variety of datasets. For each dataset, we compare our results to the results reported in the literature; hence, each table presents results of different algorithms. We note that classification is intensively explored, so the gain is bound to be small, whereas the gain for retrieval is more substantial.

4.1. Mesh classification

Given a mesh, the goal is to classify it into one of pre-defined classes. We apply our model to each mesh, as described in Section 3, where the last fully-connected layer outputs a classification prediction vector p. The three datasets utilized differ in the number of classes and the number of objects per class. We report on two evaluation measures: *Instance accuracy* is defined as the percentage of the correctly-classified objects. *Class accuracy* is defined as the mean of class instance accuracy; thus it considers all classes equally, ignoring their size. The two metrics are the same for SHREC11, which is class-balanced, and differ for the imbalanced datasets, ModelNet40 & 3D-FUTURE.

We will show below that the improvement achieved is significant for the new challenging dataset (3D-FUTURE) and is modest for the older datasets (SHREC11 & ModelNet40) that already have good results. Yet, that improvement is achieved using only $\frac{1}{8}^{th}$ of the walks.

3D-FUTURE [16]. This new dataset contains $9,992$ industrial CAD models of furniture. It consists of 7 super-categories, having 1-12 sub-categories each, for a total of 34 categories. The train/test split is $6,699$/$3,293$. This dataset is challenging both due to the objects it contains and due to its hierarchical structure, as objects in related sub-categories may resemble each other, requiring fine-grain classification.

For this dataset, we trained our model with the *class-balanced loss* of [10], which was found empirically to outperform cross-entropy. This loss handles well heavily-imbalanced datasets (the number of training shapes per category ranges between 8 to 633). It is given by:

$$CB_{softmax}(p,l) = -\frac{1-\beta}{1-\beta^{n_l}} \log \frac{e^{p_l}}{\sum_{j=1}^{C} e^{p_j}}. \quad (8)$$

For each mesh prediction vector p and label l, we weigh the cross entropy loss according to the number of training objects with the same label n_l, where β is a hyper-parameter in the range $[0,1]$, set empirically to 0.9.

Table 1 shows that our method outperforms previous methods, both point-based or multi-view. Following [16], we omit categories with less than 10 training samples from the train/test sets, thus we are left with 32 categories.

SHREC11 [32]. This dataset consists of 30 classes, each contains 20 meshes. Typical classes are camels, cats, glasses, centaurs, hands etc. Following the setup of [12], the objects in each class are split into 16 (/10) training examples and 4 (/10) testing examples.

Method	Input	Class	Instance
AttWalk (Ours)	mesh	**72.1**%	**73.7**%
MeshWalker [31]	mesh	68.9%	70.6%
PointNet++ [44]	point cloud	69.9%	-
MVCNN [48]	multi-views	69.2%	-

Table 1: **3D-FUTURE classification (class/instance accuracy).** Our results outperform those reported in [16].

Method	Input	Split-16	Split-10
AttWalk (Ours)	Mesh	**100%**	**99.7**%
PD-MeshNet [38]	Mesh	99.7%	99.1%
MeshWalker [31]	Mesh	98.6%	97.1%
HSN [55]	Mesh	-	96.1%
MeshCNN [22]	Mesh	98.6%	91.0%
GWCNN [12]	Mesh	96.6%	90.3%
SG [6]	Mesh	70.8%	62.6%

Table 2: **SHREC11 classification.** Split-16(/10) indicates that 16(/10) objects were used for training out of 20 in each class. Our method achieves perfect results for the $16/4$ split and almost perfect results for the $10/10$ split.

Table 2 compares the performance of state-of-the-art algorithms on this dataset. Each result is the average of 3 random splits into train/test sets. Our method outperforms SoTA methods. In fact, for the 16/4 split it achieves a perfect score and for the 10/10 split an almost-perfect score.

ModelNet40 [56]. This dataset contains $12,311$ CAD models from 40 categories, out of which $9,843$ models are used for training and $2,468$ for testing. Table 3 shows that our method outperforms other mesh-based methods. Thanks to our cross-walk attention that focuses on the relevant information from each walk, the good performance is achieved using only 8 walks per shape, compared to 64 walks in [31].

We note that ModelNet40 is considered challenging for mesh-based methods, since it contains many non-watertight and multiple-component object. Thus, multi-view methods outperform mesh-based methods for this particular dataset. In addition, as discussed in [49], multi-view methods rely on networks that are pre-trained not only on mesh datasets, but also on a large dataset of images (ImageNet).

4.2. Retrieval

Given a query object, the goal is to retrieve objects from a given dataset, ordered by their relevancy. Relevancy is de-

Method	Input	Class	Instance
AttWalk (Ours)	mesh	89.9%	**92.5%**
MeshWalker [31]	mesh	89.9%	92.3%
MeshNet [14]	mesh	-	91.9%
RS-CNN [35]	point cloud	-	**93.6%**
KPConv [51]	point cloud	-	92.9%
PointNet [42]	point cloud	86.2%	89.2%
Subvolume [43]	volume	-	**89.2%**
3DShapeNets [56]	volume	77.3%	84.7%
View-GCN [54]	multi-views	**96.5%**	**97.6%**
MVCNN-New [49]	multi-views	92.4%	95.0%
Rotationnet [30]	multi-views	92.4%	94.8%

Table 3: **ModelNet40 classification.** Our method achieves SoTA results compared to other mesh-based methods; comparable with MeshWalker, it does so with $\frac{1}{8}$ of the walks.

termined, for each returned object, according to the query's category and sub-category (if applied). We evaluate our method on two large-scale retrieval datasets: ModelNet40 and ShapeNet-Core55. The most common evaluation measure is the *mean average precision (mAP)*, which is used almost solely for ModelNet40. For ShapeNet-Core55, other measures are utilized as well, most notably the *Normalized Discounted Cumulative Gain (NDCG)*. Specifically, for a returned list with N objects, we consider those that belong to the query's category as positives and the others as negatives. mAP is the mean of the precision scores at every positive retrieved object position in the list. For NDCG, the relevancy of each returned object is graded between 0 to 3, considering both category and sub-category [28]. In [45], both macro and micro average results are evaluated. The macro-average gives equal weights to the scores of all the queries; the micro-average first averages the scores of each category and then averages the scores of the categories, giving every category an equal weight, regardless of its size.

ModelNet40 [56]. We use the most common $9,843/2,468$ train/test split (a few papers use other splits). Table 4 shows that our method achieves SoTA results. (The results of [31] are from our runs.)

ShapeNet-Core55. This dataset, which is a subset of ShapeNet, contains $51,162$ 3D objects from 55 categories, each is subdivided into 1-28 sub-categories. The dataset is split into $35,764$ / $5,133$ / $10,265$ training / validation / testing objects. The results are reported on the test set, using the evaluation code provided by [45]. As in [45], we retrieve up to 1000 object whose Euclidean distance from the prediction vector of the query object is smaller than $2m$, where m is the margin hyperparameter from Equation 6.

Table 5 shows that our performance outperforms those of SoTA methods in both metrics. As NDCG takes into account the sub-categories, our high scores demonstrate how well our cross-walk attention captures the objects' distinctive features. This property is also observed in Figure 1. The bathtub and the desk are erroneously classified by [31] as a lamp and a sink, respectively. Our method correctly focuses on walks that depict a broader view of the objects, including their outlines, resulting in correct classification.

Method	Input	mAP
AttWalk	Mesh	**91.2**
MeshWalker [31]	Mesh	87.7
MeshNet [14]	Mesh	81.9
GWCNN [12]	Mesh	59.0
DensePoint [34]	Point Cloud	**88.5**
MVCNN [48]	multi-views	79.5
SeqViews [21]	multi-views	**89.1**

Table 4: **ModelNet40 retrieval.** Our method outperforms other methods applied to the full dataset.

	microAll		macroAll	
Method	mAP	NDCG	mAP	NDCG
AttWalk	**81.1**	**86.7**	**65.5**	**68.2**
DLAN [45]	66.3	76.2	47.7	56.3
ViewGCN [54]	**78.4**	85.2	**60.2**	**66.5**
GIFT [3]	64.0	76.5	44.7	54.8
MVCNN [48]	73.5	81.5	56.6	64.0
RotationNet [30]	77.2	**86.5**	58.3	65.6

Table 5: **ShapeNet-Core55 retrieval [45].** Our method outperforms both mesh-based (upper table) and multi-view (lower table) methods.

5. Ablation Study

Insights on the most/least attentive walks. What characterizes attentive walks? To answer this question we analyze the results of ModelNet40 classification. We consider the attention rank of a walk according to its contribution to the final mesh feature vector F_a. Since the contribution of a walk to each entry of F_a differs, we average these contributions. Hereafter we discuss the affects we observed.

(1) All walks contribute to the final feature vector, however the contribution of the most attentive walk is 70% higher than that of the least attentive walk (17% vs. 10%).

(2) The most-attentive walk is 36% longer than the least attentive walk on average. A possible explanation is that longer edges tend to describe the outline of the object, which is meaningful for capturing the shape of the object.

(3) No correlation is found between the the number of faces adjacent to a walk and the attentiveness of this walk. How-

ever, similarly to (2), the most attentive walks "cover" more surface area (of the faces adjacent to the walk).

(4) The median Gaussian curvature of the most attentive walk is smaller (by 21%) than that of the least attentive walk. This can be explained, similarly to (2), by the fact that attentive walks tend to describe the major parts of the object and not to focus on small (high-curvature) details.

Training: 2-phase vs. 1-phase. Recall that we train our model in two phases, first training only MeshWalker and then training only the attention module. We compare this strategy to a single-phase (end-to-end) training, for ModelNet40 and for 3D-Future. For both datasets, 2-phase training yields better results: The instance accuracy is 92.5 vs. 89.2 for ModelNet40 and 73.7 vs. 71.4 for 3D-Future. A possible explanation is that 2-phase training forces MeshWalker to learn the most meaningful features for each walk independently, whereas an end-to-end system makes it easier to focus on the "easier" (more meaningful) walks. But, as we saw above, even the least-attentive walks contribute to the final results and hence should not be neglected.

Number of walks. How many random walks suffice for optimal exploration of a mesh? Table 6 shows that 8 walks (in training and testing) already achieve the best results for SHREC11 classification. These results are reinforced in Table 3 for ModelNet40.

# Walks	1	2	4	8	16	32
Accuracy	98.1	99.1	99.6	99.7	99.7	99.7

Table 6: **Number of walks.** 8 walks suffice for best performance, in contrast to [31], where 32-64 walks are used.

Alternative walk aggregation methods. Our model aggregates walks using our novel attention scheme. Table 7 compares our results to alternative aggregation strategies for generating a single shape descriptor from multiple walk features. It shows that our proposed scheme outperforms average & max pooling on f_{w_i}, as well as adding average & max pooling to the self-attention matrix H_a. The latter two aggregations demonstrate that indeed self-attention by itself does not suffice for achieving SoTA results.

Walk length. The longer the walk, the better the performance. However, once the walk reaches 0.3 of the vertices, the performance does not improve further. For instance, on SHREC11, we get 90.5 accuracy when the walk contains 0.1 of the vertices, 99.2 accuracy for 0.2 of the vertices, and 99.7 accuracy for 0.3 or more of the vertices.

Affect of the TCL loss (Eq. 7). If we used the loss of Eq. 5 instead of Eq. 7 in retrieval, the mAP would be as follows: (1) For ModelNet40 91.0 compared to 91.2 (still SoTA); (2) For ShapeNetCore55 microAll 78.2 compared to 81.1 (comparable to previous SoTA); (3) For ShapeNetCore55 macroAll 63.5 compared to 65.5 (still SoTA).

Aggregation method	Class	Instance
Cross-walk Attention (our)	**72.1**	**73.7**
Average pooling	70.1	71.0
Max pooling	58.2	60.5
H_a + Average pooling	69.3	71.7
H_a + Max pooling	69.7	71.3

Table 7: **Walk aggregation approaches.** Our attention outperforms other aggregation strategies (3D-Future).

Runtime. The average inference time on Quadro P6000 is 62.2 milliseconds (on ModelNet40) (compared to 128 ms in [31]). Training takes 94.3 milliseconds per object.

Limitations. Figure 5 illustrates a failure case, where a majority rule of [31] would be preferable. Though 5 of 8 walks indicate that the shape is a bathtub, our attention gives more weight to features that indicate that this is a bed. The most attentive walk (in cyan), which provides a global view of the shape, classifies it as a bed due to the special shape of this bathtub. The least attentive walk (in magenta) visits mostly the tap and thus classifies the bathtub as a sink.

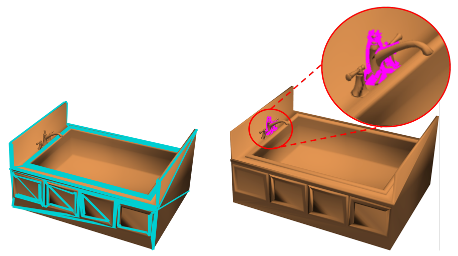

Figure 5: **Limitation.** Our algorithm classifies the bathtub as a bed, giving more attention to features resembling a bed, as shown by the most attentive walk (in cyan).

6. Conclusion

This paper introduced attention into a 3D learning framework, which is under-explored. It showed how multiple random walks along the surface may jointly indicate the most attentive features of a 3D mesh. The key idea is that exploring the mesh in different ways, by different walks, can be leveraged for both learning the meaningful attributes of the surface and to reduce the number of walks needed. Our approach achieves state-of-the-art results for shape classification and shape retrieval on commonly-used datasets.

We intend to adapt our approach to other applications, most notably shape segmentation. Adjusting our approach to multi-scale is also a direction that worth further studying.

Acknowledgements. We gratefully acknowledge the support of the Israel Science Foundation 1083/18, the Ministry Science and Technology (MOST) 3-17513, and Rafael.

References

[1] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM Transactions on Graphics (TOG)*, 2018.

[2] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[3] Song Bai, Xiang Bai, Zhichao Zhou, Zhaoxiang Zhang, and Longin Jan Latecki. Gift: A real-time and scalable 3d shape search engine. In *CVPR*, 2016.

[4] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. 3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks. *IEEE Robotics and Automation Letters*, 3:3145–3152, 2018.

[5] Davide Boscaini, Jonathan Masci, Emanuele Rodoià, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *NeurIPS*, 2016.

[6] Alexander M Bronstein, Michael M Bronstein, Leonidas J Guibas, and Maks Ovsjanikov. Shape google: Geometric words and expressions for invariant shape retrieval. *ACM Transactions on Graphics (TOG)*, 2011.

[7] Songle Chen, Lintao Zheng, Yan Zhang, Zhixin Sun, and Kai Xu. Veram: View-enhanced recurrent attention model for 3D shape classification. *IEEE transactions on visualization and computer graphics*, 25(12):3244–3257, 2018.

[8] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. In *EMNLP*, 2016.

[9] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

[10] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In *CVPR*, 2019.

[11] 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.

[12] Danielle Ezuz, Justin Solomon, Vladimir G Kim, and Mirela Ben-Chen. GWCNN: A metric alignment layer for deep shape analysis. In *Computer Graphics Forum*, 2017.

[13] Gabriele Fanelli, Thibaut Weise, Juergen Gall, and Luc Van Gool. Real time head pose estimation from consumer depth cameras. In *ICPR*, 2011.

[14] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. Meshnet: Mesh neural network for 3D shape representation. In *AAAI*, volume 33, pages 8279–8286, 2019.

[15] Yifan Feng, Zizhao Zhang, Xibin Zhao, Rongrong Ji, and Yue Gao. GVCNN: group-view convolutional neural networks for 3D shape recognition. In *CVPR*, 2018.

[16] Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 3D-FUTURE: 3D Furniture shape with TextURE. *arXiv preprint arXiv:2009.09633*, 2020.

[17] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3D shape. In *CVPR*, 2020.

[18] Abubakar Sulaiman Gezawa, Yan Zhang, Qicong Wang, and Lei Yunqi. A review on deep learning approaches for 3D data representations in retrieval and classifications. *IEEE Access*, 8:57566–57593, 2020.

[19] Shunwang Gong, Lei Chen, Michael Bronstein, and Stefanos Zafeiriou. SpiralNet++: A fast and highly efficient mesh convolution operator. In *ICCV Workshops*, 2019.

[20] Zhizhong Han, Honglei Lu, Zhenbao Liu, Chi-Man Vong, Yu-Shen Liu, Matthias Zwicker, Junwei Han, and CL Philip Chen. 3D2SeqViews: Aggregating sequential views for 3D global feature learning by CNN with hierarchical attention aggregation. *IEEE Transactions on Image Processing*, 28(8):3986–3999, 2019.

[21] Zhizhong Han, Mingyang Shang, Zhenbao Liu, Chi-Man Vong, Yu-Shen Liu, Matthias Zwicker, Junwei Han, and CL Philip Chen. Seqviews2seqlabels: Learning 3D global features via aggregating sequential views by rnn with attention. *IEEE Transactions on Image Processing*, 28(2):658–672, 2018.

[22] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. MeshCNN: a network with an edge. *ACM Transactions on Graphics (TOG)*, 2019.

[23] Xinwei He, Tengteng Huang, Song Bai, and Xiang Bai. View n-gram network for 3D object retrieval. In *ICCV*, 2019.

[24] Xinwei He, Yang Zhou, Zhichao Zhou, Song Bai, and Xiang Bai. Triplet-center loss for multi-view 3D object retrieval. In *CVPR*, 2018.

[25] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[26] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. In *ICCV*, 2019.

[27] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *CVPR*, 2020.

[28] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 2002.

[29] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. Local implicit grid representations for 3D scenes. In *CVPR*, 2020.

[30] Asako Kanezaki, Yasuyuki Matsushita, and Yoshifumi Nishida. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In *CVPR*, 2018.

[31] Alon Lahav and Ayellet Tal. Meshwalker: Deep mesh understanding by random walks. *ACM Transactions on Graphics (TOG)*, 2020.

[32] Z Lian, A Godil, B Bustos, M Daoudi, J Hermans, S Kawamura, Y Kurita, G Lavoua, and P Dp Suetens. Shape retrieval on non-rigid 3D watertight meshes. In *3DOR*, 2011.

[33] Kevin Lin, Lijuan Wang, and Zicheng Liu. End-to-end human pose and mesh reconstruction with transformers. *arXiv preprint arXiv:2012.09760*, 2020.

[34] Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. Densepoint: Learning densely contextual representation for efficient point cloud processing. In *ICCV*, 2019.

[35] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *CVPR*, 2019.

[36] Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G Kim, and Yaron Lipman. Convolutional neural networks on surfaces via seamless toric covers. *ACM Transactions on Graphics (TOG)*, 2017.

[37] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*, 2019.

[38] Francesco Milano, Antonio Loquercio, Antoni Rosinol, Davide Scaramuzza, and Luca Carlone. Primal-dual mesh convolutional neural networks. In *NeurIPS*, 2020.

[39] Anshul Paigwar, Ozgur Erkent, Christian Wolf, and Christian Laugier. Attentional pointnet for 3D-object detection in point clouds. In *CVPR*, 2019.

[40] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019.

[41] Adrien Poulenard and Maks Ovsjanikov. Multi-directional geodesic neural networks via equivariant convolution. *ACM Transactions on Graphics (TOG)*, 2018.

[42] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. In *CVPR*, 2017.

[43] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3D data. In *CVPR*, 2016.

[44] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *NeurIPS*, 2017.

[45] Manolis Savva, Fisher Yu, Hao Su, Asako Kanezaki, Takahiko Furuya, Ryutarou Ohbuchi, Zhichao Zhou, Rui Yu, Song Bai, Xiang Bai, et al. Large-scale 3D shape retrieval from ShapeNet Core55: SHREC'17 track. In *3DOR*, 2017.

[46] Jonas Schult, Francis Engelmann, Theodora Kontogianni, and Bastian Leibe. Dualconvmesh-net: Joint geodesic and euclidean convolutions on 3D meshes. In *CVPR*, 2020.

[47] Ayan Sinha, Jing Bai, and Karthik Ramani. Deep learning 3D shape surfaces using geometry images. In *ECCV*, 2016.

[48] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *ICCV*, 2015.

[49] Jong-Chyi Su, Matheus Gadelha, Rui Wang, and Subhransu Maji. A deeper look at 3D shape classifiers. In *ECCV*, 2018.

[50] Weiwei Sun, Wei Jiang, Eduard Trulls, Andrea Tagliasacchi, and Kwang Moo Yi. Acne: Attentive context normalization for robust permutation-equivariant learning. In *CVPR*, 2020.

[51] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. KPConv: Flexible and deformable convolution for point clouds. In *ICCV*, 2019.

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

[53] Nitika Verma, Edmond Boyer, and Jakob Verbeek. Feastnet: Feature-steered graph convolutions for 3D shape analysis. In *CVPR*, 2018.

[54] Xin Wei, Ruixuan Yu, and Jian Sun. View-gcn: View-based graph convolutional network for 3d shape analysis. In *CVPR*, 2020.

[55] Ruben Wiersma, Elmar Eisemann, and Klaus Hildebrandt. Cnns on surfaces using rotation-equivariant features. *ACM Transactions on Graphics (TOG)*, 2020.

[56] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D shapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015.

[57] Qian Xie, Yu-Kun Lai, Jing Wu, Zhoutao Wang, Yiming Zhang, Kai Xu, and Jun Wang. Mlcvnet: Multi-level context votenet for 3D object detection. In *CVPR*, 2020.

[58] Saining Xie, Sainan Liu, Zeyu Chen, and Zhuowen Tu. Attentional shapecontextnet for point cloud recognition. In *CVPR*, 2018.

[59] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

[60] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *ECCV*, 2018.

[61] Jie Yang, Lin Gao, Qingyang Tan, Yihua Huang, Shihong Xia, and Yu-Kun Lai. Multiscale mesh deformation component analysis with attention-based autoencoders. *arXiv preprint arXiv:2012.02459*, 2020.

[62] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point clouds with self-attention and gumbel subset sampling. In *CVPR*, 2019.

[63] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. In *CVPR*, 2016.

[64] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *ICML*, 2019.

[65] Wenxiao Zhang and Chunxia Xiao. Pcan: 3D attention map learning using contextual information for point cloud based retrieval. In *CVPR*, 2019.

[66] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. In *CVPR*, 2020.