UV-Net: Learning from Boundary Representations

Pradeep Kumar Jayaraman
Autodesk Research

Aditya Sanghi
Autodesk Research

Joseph G. Lambourne
Autodesk Research

Karl D.D. Willis
Autodesk Research

Thomas Davies
Autodesk

Hooman Shayani
Autodesk Research

Nigel Morris
Autodesk Research

Abstract

We introduce UV-Net, a novel neural network architecture and representation designed to operate directly on Boundary representation (B-rep) data from 3D CAD models. The B-rep format is widely used in the design, simulation and manufacturing industries to enable sophisticated and precise CAD modeling operations. However, B-rep data presents some unique challenges when used with modern machine learning due to the complexity of the data structure and its support for both continuous non-Euclidean geometric entities and discrete topological entities. In this paper, we propose a unified representation for B-rep data that exploits the U and V parameter domain of curves and surfaces to model geometry, and an adjacency graph to explicitly model topology. This leads to a unique and efficient network architecture, UV-Net, that couples image and graph convolutional neural networks in a compute and memory-efficient manner. To aid in future research we present a synthetic labelled B-rep dataset, SolidLetters, derived from human designed fonts with variations in both geometry and topology. Finally we demonstrate that UV-Net can generalize to supervised and unsupervised tasks on five datasets, while outperforming alternate 3D shape representations such as point clouds, voxels, and meshes.

1. Introduction

Parametric curves and surfaces form the basis of computer-aided design (CAD) and are widely used in design, simulation, and manufacturing. CAD software is primarily concerned with modeling and representing 3D solids—closed, watertight shapes which describe objects unambiguously with consistently oriented patches of surface geometry. The industry-wide standard to represent solid models is the Boundary representation (B-rep) [41, 23]. The B-rep is a versatile data structure comprised of faces (bounded portions of surfaces), edges (bounded pieces of curves) and vertices (points), glued together with topological connections between them. The B-rep enables a variety of parametric
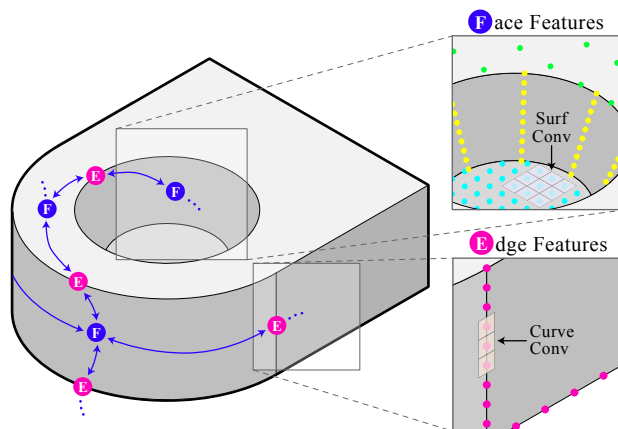

Figure 1: UV-Net builds features by sampling points on the edges and faces of solid models. These features are then message-passed among adjacent topological entities.

curves and surfaces, such as lines, arcs, planes, cylinders, toruses and Non-Uniform Rational B-Splines (NURBS), to precisely represent complex 3D shapes formed from CAD modeling operations such as extrusions, fillets, and Booleans. CAD users interact directly with B-rep faces, edges, and vertices to select, align, and modify 3D shapes. To leverage the recent advances of deep neural networks in CAD software, an appropriate representation of B-rep data is required. Such a representation has the potential to unlock numerous CAD applications such as auto-complete of modeling operations, smart selection tools, shape similarity search and many more. Critical to enabling these applications is a representation that encodes the B-rep entities themselves.

Despite widespread usage of B-rep data in the industry, there exists limited research on applying deep neural networks to this representation directly. There are numerous challenges in feeding B-rep data to neural networks. B-rep data consists of disparate geometric and topological entities, such as parametric curves and surfaces, each with their own set of parameters. Moreover, the mapping between a shape and a surface type is not one-to-one, for example, a plane

can be represented as a B-spline of arbitrary degree. This means raw surface information, such as parametric coefficients or spline control points, cannot be fed directly into a neural network, as it would not be invariant to the specific parameterization. Finally, consideration must be given to how different curve and surface geometry are connected to form the entire shape, i.e. the topology.

An alternate approach is to preprocess B-rep data into well-studied representations, such as images, voxels, point clouds, or triangle meshes. Although plausible, such conversions are neither differentiable, nor trivial. Discretized representations, such as point clouds or voxels, suffer from loss of fidelity and may lose the critical mapping back to the original B-rep entities. Conversion to triangle meshes can be non-trivial and prone to failure when high quality, manifold meshes are required [13].

To tackle these challenges, we present UV-Net, a novel neural network architecture and representation designed to operate directly on B-rep entities (Figure 1). In this paper, we make the following contributions:

- We present a new representation of 3D CAD models derived from B-reps, which captures geometric features from the parameter domain as a regular grid, and topological information as a graph.

- We propose a novel architecture which couples an image CNN and a hierarchical graph-neural network in a compute and memory-efficient manner.

- We create and release a synthetic labeled dataset: SolidLetters, which unlike other synthetic datasets, is balanced, and has variations in both geometry and topology.

- We demonstrate the efficacy of UV-Net on multiple tasks including 3D shape classification, segmentation, and self-supervised shape retrieval on unlabeled data. We achieve state-of-the-art results on both classification and segmentation tasks by leveraging the full B-rep data structure.

2. Related Work

Common geometric representations Modern neural networks work with several discrete 3D representations like point clouds, voxels, meshes and multi-view images. A B-rep can be easily sampled to obtain point clouds [30, 40]. A problem with this conversion is that CAD models often contain small features that convey important information. A prohibitively dense point cloud might be required to capture such fine details. 3D CNNs [48] can be applied to voxelized B-reps, as shown by Zhang et al. [46] for classification. Unfortunately, there is a cubic compute and memory cost to increasing the voxel grid resolution to capture small faces from a B-rep. O-CNN [38] can alleviate this problem using sparse octrees, however, very deep octrees may

be needed to delineate tiny faces common in B-rep data. Neural networks representing signed distance [27, 9] or occupancy [25] functions are grid-free and concise, but need to learn mappings to B-rep face and edges along with positional encodings to support downstream applications, which is a challenging problem. Triangle meshes on the other hand better preserve the geometric and topological information of a solid model [16, 43]. These methods require the B-rep to be converted to watertight, manifold meshes with tight constraints on vertex/edge count, edge length, and angles; a difficult task prone to failure [18, 13]. Finally, multi-view images represent 3D shapes by rendering, and have shown excellent results on shape classification and retrieval [36]. Such renderings are not expressive enough to represent and map back to the multitude of entities in a B-rep, thereby limiting applications. There is a recent interest in the machine learning community in the generation of parametric geometry such as Bézier curves [24, 39, 35], splines [12], Coons patches [34] and binary space partitioning planes [8]. However, these methods do not deal with feature extraction from B-reps with disparate parametric curve/surface types.

3D geometry as images Closely related to our consideration of geometry as regular grids or images are geometry images [15] where arbitrary meshes were parametrized into 2D grids for compression and resampling. Sinha et al. [33] parametrized meshes globally as images to apply CNNs. Groueix et al. [14] learnt the parametrization of point clouds by deforming grids of points. Kawasaki et al. [20] smoothed the normal map of B-spline surfaces for fairing. Different from these works, we deal with parametric shapes employed in CAD, and focus on deriving a representation from the B-rep while retaining geometric and topological information.

Boundary representations Few neural networks are capable of directly consuming B-rep data. Initial attempts before the deep learning era focused on automatically recognizing machining features in a solid model. These methods convert the B-rep into a face-adjacency graph [2], with features such as surface type and edge convexity, that is then used by rule-based schemes [19] or simple neural networks [29, 26]. However, hand-engineered features struggle to generalize well to other tasks. Babic et al. [4] surveys several classic machine learning methods for feature classification on B-reps. Very recently, Cao et al. [6] used a graph neural network to segment the faces of a B-rep by converting it into a face-adjacency graph. A major limitation of this method is that it can only work on B-reps with planar faces, as it uses the coefficients of the scalar plane equation as node features to describe the geometry. In contrast to these works, we aim to derive a general representation from the B-rep that is suitable for a wide range of tasks, and can leverage advances in modern deep learning methods.
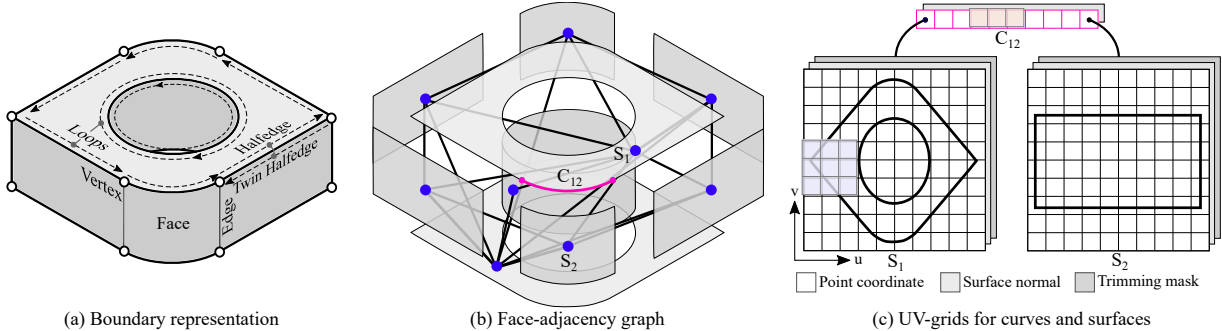
Figure 2: Our representation. (a) The B-rep is a complex data structure with several geometric and topological entities that is difficult to feed to neural networks. (b) We derive a face-adjacency graph from the B-rep to capture topological information. (c) Each face and edge in a B-rep contains a parametric surface and curve, respectively, which we represent as regular UV-grids, and store as node and edge attributes in the graph. Local neighborhoods in UV-grids map to local regions in the geometry.

3. Method

In this section we review the B-rep data structure, and introduce UV-Net's representation and network architecture.

3.1. Input representation

The B-rep data structure comprises several topological entities—faces, edges, halfedges, and vertices, with connections between them, see Figure 2 (a). Faces are the visible portion of parametric surfaces such as planes, cones, cylinders, toruses, and splines. Edges are the visible interval of parametric curves and vertices are the endpoints of edges. Each face is delimited by one or more loops of halfedges. Anti-clockwise loops define outer boundaries while clockwise loops define internal holes. Solid modeling packages are designed to generate closed and watertight B-rep models, in which every edge contains two halfedges on adjacent faces. The data structure also stores many references allowing efficient navigation between all adjacent entities [23].

Although expressive, the B-rep is a complex data structure and is difficult to feed to neural networks in its original form. Our goal is to extract the most informative geometric and topological information from the B-rep, and convert it into a representation that can easily and efficiently work with existing neural network architectures.

Topology UV-Net uses a face-adjacency graph derived from the B-rep, $G(V, E)$, to model the topology where the vertices V represent the faces in the B-rep, while the edges E encode the connectivity between the faces, as shown in Figure 2 (b). This can be easily built in constant time complexity by traversing through the halfedges of the B-rep: current face $\rightarrow$ halfedges $\rightarrow$ twin-halfedges $\rightarrow$ neighboring faces. The face adjacency captures the two most geometrically and topologically rich entities: faces and edges from the B-rep, and is sufficient to capture both local and global information about a solid, as we later demonstrate.

Curve geometry Each topological edge in a B-rep has an associated parametric curve to define the actual geometry. Consider one such parametric curve $\mathbf{C}(u)$, which is a map from an interval $[u_{\min}, u_{\max}] \in \mathbb{R}$, the parameter domain, to the geometry domain $\mathbb{R}^3$. The curve could be parameterized as a line, circular arc, or B-spline; we only expect that an interface is available to evaluate the curve and optionally, its first order derivative. Our idea is to represent the geometry of the curve by discretizing its parameter domain [20] as a regular 1D grid by a uniform step size $\delta u = \frac{u_{\max} - u_{\min}}{M-1}$, where M is the number of chosen samples, as shown in Figure 2 (c). At each of the discretized points in the parameter domain u_k, we can attach a set of features evaluated from the curve, e.g., absolute point coordinates $\mathbf{C}(u_k)$, and optionally the unit tangent vector $\hat{\mathbf{C}}_u(u_k)$ as features. This 1D UV-grid is set as input edge features in G as shown in Figure 2(c).

Surface geometry Each topological face in a B-rep has an associated surface geometry that can be a plane, sphere, cylinder, cone, or a freeform NURBS surface. The surfaces are trimmed by the halfedge loops that run along the boundary of the face to expose only a portion of the surface as a visible region. Consider one such parametric surface $\mathbf{S}(u, v)$ which is a map from a 2D interval $[u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}] \in \mathbb{R}^2$, the parameter domain, to the geometry domain $\mathbb{R}^3$. We discretize the parameter domain into a regular 2D grid of samples with step sizes $\delta u = \frac{u_{\max} - u_{\min}}{M-1}$, and $\delta v = \frac{v_{\max} - v_{\min}}{N-1}$, where M and N are the number of samples along each dimension, as shown in Figure 2 (c). The intervals $[u_{\min}, u_{\max}]$ and $[v_{\min}, v_{\max}]$ are chosen such that they closely bound the loop that defines the visible region. At each of these grid points indexed by (k, l), we attach the following local features encoding the geometry of the surface as channels: (1) 3D absolute point position $\mathbf{S}(u_k, v_l)$ (the scale of the solid is normalized into a cube of size 2 and centered at origin). (2) Optionally, the 3D absolute surface normal $\frac{\mathbf{S}_u(u_k, v_l) \times \mathbf{S}_v(u_k, v_l)}{\|\mathbf{S}_u(u_k, v_l) \times \mathbf{S}_v(u_k, v_l)\|}$ pointing outwards
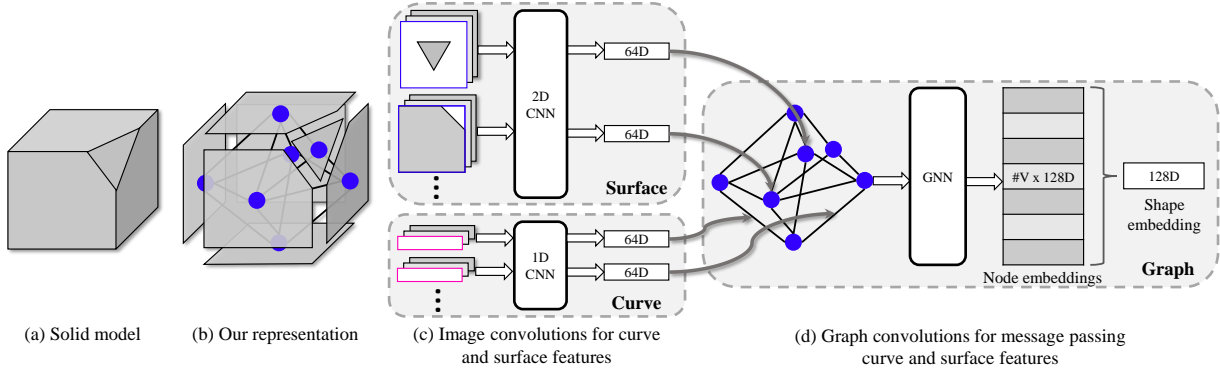
Figure 3: UV-Net encoder architecture. (a) A solid model is represented by (b) a set of regular UV-grids representing each face's and edge's geometry by discretizing the parameter domain, and a graph that captures its topology with face-adjacency information. (c) Curve and surface features are extracted from the UV-grids with 1D and 2D CNNs, respectively. (d) These features are treated as edge and node embeddings of the graph and further processed by graph convolutions. The result is a set of node embeddings, that can be pooled to get the shape embedding of the solid model.

consistently. (3) Trimming mask with 1 and 0 representing samples that are in the visible region and trimmed region, respectively. This 2D UV-grid is defined as input node features in G. The representation is flexible enough to incorporate other features like curvature based on the downstream task.

We set the number of samples $M=N=10$ in all experiments throughout the paper. This is not a technical restriction, rather it is convenient to form mini-batches of features. A fixed step size is sufficient when the mapping between parameter and geometry domains are roughly uniform. We quantitatively evaluate this in the supplementary material. In the case of extreme parameterizations with high stretching, it is possible to derive a step size to upper bound the distance between samples [47].

Advantages The UV-Net representation has several advantages: (1) Evaluating curves/surfaces at a set of parameters is fast for both primitive and spline surfaces [32]. (2) The representation is sparse and scales with the number curves and surfaces in the B-rep. (3) The grid is largely invariant to the exact parametrization. For example, the grid does not change when a planar surface is converted into a NURBS patch, or when degree elevation or knot insertion is performed, since the parameterization and geometry remain identical [28]. In contrast, the raw curve/surface equation will change significantly. (4) Finally, local neighborhoods in the parameter domain (UV-grids) correspond to local neighborhoods in curve/surface geometry domain, hence hierarchical feature extraction on the manifold [5] is possible.

3.2. Network architecture

With this representation, we first perform image convolutions on the curve and surface UV-grids. These local curve/surface features are then propagated over the entire B-rep with graph convolutions as shown in Figure 3.

Curve & surface convolution Our surface CNN takes in 2D UV-grids with typically 4 or 7 channels (3 xyz, 3 normals, 1 trimming mask) and is defined as: $\text{Conv}(4/7, 64, 3) \rightarrow \text{Conv}(64, 128, 3) \rightarrow \text{Conv}(128, 256, 3) \rightarrow \text{Pool}(1, 1) \rightarrow \text{FC}(256, 64)$, where $\text{Conv}(i, o, k)$ is an image convolutional layer with i input channels, o output channels, and kernel size k, $\text{Pool}(n, n)$ is an adaptive average pooling layer which outputs a $n \times n$ feature map, and $\text{FC}(i, o)$ is a fully connected layer which takes an input in i-D vector and maps it to o-D vector. Our curve CNN takes 1D UV-grids computed from the curves lying in the edges of the B-rep, and is defined similarly with 1D convolutional and pooling layers. The weights of the curve and surface CNN are shared among all edges and faces in a B-rep, respectively, making them permutation-invariant. Both convolutional and fully-connected layers do not have biases, and include batch normalization and the LeakyReLU activation function. We pad the features with size $\lfloor k/2 \rfloor$ to retain the spatial dimensions of the input.

Message passing The output of curve and surface CNNs are hidden features treated as input edge and node features to the graph neural network. Given the initial features, we compute the hidden node features $h_v^{(k)}$ in graph layer $k \in 1 \ldots K$, by aggregating the input node features $h_v^{(k-1)}$ from a one-hop neighborhood $u \in N(v)$ while conditioning them on the edge features $h_{uv}^{(k-1)}$:

$$h_v^{(k)} = \phi^{(k)} \bigg((1 + \epsilon^{(k)}) \, h_v^{(k-1)} + \sum_{u \in N(v)} f_\Theta \big(h_{uv}^{(k-1)} \big) \odot h_u^{(k-1)} \bigg), \quad (1)$$

where $\phi^{(k)}$ is a multi-layer perceptron (MLP) with two fully connected layers $FC(64, 64) \rightarrow FC(64, 64)$, $\epsilon^{(k)}$ is a learn-

able parameter to distinguish the center nodes from the neighbors and f_Θ is a linear projection from the edge to node feature space. This update equation extends the Graph Isomorphism Network [45], with additional consideration of edge features. The hidden edge features are next updated similarly while considering the features of the endpoint nodes:

$$h_{uv}^{(k)} = \psi^{(k)}\left((1 + \gamma^{(k)})\, h_{uv}^{(k-1)} + f_\Xi\big(h_u^{(k-1)} + h_v^{(k-1)}\big)\right),$$
(2)

where $\psi^{(k)}$ is a 2-layer MLP as before, $\gamma^{(k)}$ is a learnable parameter to distinguish the edge features from its neighbors, and f_Ξ is a linear projection from the node to edge feature space. At the end, we then take all the hidden node features $\{h_v^{(k)} \mid k \in 1 \dots K\}$ and apply an element-wise max-pooling operation across the nodes to obtain hierarchical graph-level feature vectors from every layer $\{h^{(k)} \mid k \in 1 \dots K\}$, where $h^{(k)} = \text{maxpool}_{v \in V}(h_v^{(k)})$. These features are then linearly projected into 128D vectors and summed to obtain the final shape embedding:

$$h_G = \sum_{k=1}^{K} w^{(k)} \cdot h^{(k)} + b^{(k)}.$$
(3)

We use $K = 2$ graph layers in all experiments. The node and graph embeddings obtained from the network can be used for several downstream applications as detailed next.

4. Experiments

In this section, we qualitatively and quantitatively evaluate UV-Net on 3D shape classification, segmentation, and shape retrieval on unlabelled data.

4.1. Datasets

We briefly introduce the five datasets used in our experiments and provide further details in the supplementary material. We select the datasets below as they are available in B-rep format, unlike many common benchmark datasets provided in mesh format.

Machining feature dataset [46] a synthetic labeled, balanced dataset representing machining features such as chamfers and circular end pockets applied to a cube. It has 23,995 3D shapes (∼1000 per class) split into 24 classes.

MFCAD dataset [6] a synthetic segmentation dataset of 15,488 3D shapes, similar to the Machining feature dataset, but with multiple machining features. 16 different segmentation labels are used and applied per face.

FabWave dataset [1] a small labeled, imbalanced collection of 5,373 3D shapes split into 52 mechanical part classes, such as brackets, gears, and o-rings.

ABC dataset [22] a real-world collection of millions of 3D shapes. The dataset is unlabelled, imbalanced, and has many duplicates. We remove duplicates and use a subset of 46k models in our experiments.

SolidLetters dataset our dataset consists of 96k 3D shapes generated by randomly extruding and filleting the 26 alphabet letters (a–z) to form class categories across 2002 style categories from fonts. Compared to other synthetic datasets that have similar intra-class topology, SolidLetters contains significant variations in both geometry and topology, due to font variety, and is well-balanced.

4.2. Tasks

We now compare UV-Net to PointNet [30], DGCNN [40], and MeshCNN [16] on several standard tasks. We show additional results from the baseline methods presented with the Machining feature and MFCAD datasets.

4.2.1 Classification

We first evaluate our method on the task of 3D shape classification. The ability to classify 3D components in large B-rep assemblies is valuable for numerous applications including product lifecycle management and automation of repetitive tasks such as simulation setup. We show the advantages of using both geometry and topology in the B-rep. This is particularly important in datasets where data within a class has high geometric variance but similar topology, as is common in parametric CAD modeling. Our network comprises the UV-Net encoder network in Figure 3 followed by a non-linear classifier (2-layer MLP) that maps the 128D shape embedding into class logits. Our input geometric features include xyz coordinates and the trimming mask.

We train point cloud-based methods on 2048 points sampled uniformly from the solid model, FeatureNet [46], the baseline for the Machining feature dataset, on 64^3 voxel grids, and MeshCNN [16] on triangle meshes. For the Machining Feature dataset we convert B-reps into high-quality, watertight, manifold meshes as required by MeshCNN using the finite-element mesher in Autodesk Fusion 360 with a target edge-count of 2000 edges. As MeshCNN requires all meshes to have a similar edge count, we find it is impractical to use with datasets of varying shape complexity, such as FabWave, SolidLetters, and ABC. Although it may be feasible to use a target edge count suitable for the most complex shape in the dataset, in practice this dramatically increases training time and limits the advantages of mesh pooling.

We train all models to a maximum of 350 epochs with cross-entropy loss and the Adam [21] optimizer. Table 1 shows that our method achieves the best classification accuracy on all datasets. Unstructured representations suffer when data within a class has high geometric variance but similar topology, since they cannot model the latter explicitly.

Table 1: Solid model classification.

Dataset	Model	Accuracy (%)	#Param.
Machining Feature	UV-Net	**99.94 ± 0.00**	1.34M
	PointNet (2048)	87.13 ± 0.15	0.81M
	DGCNN (2048)	92.81 ± 0.69	1.81M
	FeatureNet (64^3)	98.85 ± 0.48	33.94M
	MeshCNN (2000)	98.90 ± 0.70	0.67M
FabWave	UV-Net	**94.51 ± 0.10**	1.35M
	PointNet (2048)	80.08 ± 3.61	0.82M
	DGCNN (2048)	69.95 ± 2.37	1.81M
SolidLetters	UV-Net	**97.24 ± 0.10**	1.34M
	PointNet (2048)	94.72 ± 0.17	0.81M
	DGCNN (2048)	96.62 ± 0.13	1.81M

Notably, we outperform FeatureNet [46] on their dataset, and obtain the highest results on SolidLetters, demonstrating that our method can exploit both geometry and topology.

4.2.2 Segmentation

We now consider the problem of segmenting the faces of a B-rep, a classic task with applications in machining feature recognition, computer-aided process planning and CAD modeling history reconstruction. We consider the MFCAD and ABC datasets in this experiment and demonstrate the benefit of directly working with B-rep entities. To work around lack of labels in the ABC dataset, we use the Autodesk Shape Manager [3], a commercial solid-modeling kernel, to assign labels indicating the CAD operation likely to have created the face, such as *ExtrudeSide*, *ExtrudeEnd*, or *Fillet*, we provide more details in supplementary material.

Our segmentation network is similar to the classification network with a small difference: we concatenate the shape embedding to each of the node embeddings, and use a nonlinear classifier to output per-node logits. We additionally include curve tangents and surface normals in the edge and face input features, respectively.

To investigate the benefits of working with B-rep data directly, we compare against established point and mesh-based methods. We mesh the B-reps in the MFCAD dataset, as previously described and discard 27 shapes that fail to mesh. To generate point clouds for both the MFCAD and ABC datasets, we first convert the B-reps into render-meshes, i.e., non-watertight, disjoint meshes. We then sample the triangles uniformly based on the surface area to generate 2048 points. The mapping between the faces and primitives (edges, and points) are retained, so that we can perform a per-face voting to compute face-level scores.

We train the models as before using the cross-entropy loss. Considering each face in the B-rep as a data point, we report the accuracy, per-class accuracy and intersection-over-union (IoU) metrics in Table 2. Results show that UV-Net solves the face segmentation problem in the MFCAD dataset, outperforming their baseline method [6] and point cloud-based methods by a wide margin. MeshCNN [16] obtains very similar results to UV-Net; we suggest this is due to the dihedral angle feature used, which many segmentation labels in the MFCAD dataset strongly correlate with. Our method achieves state-of-the-art results while operating on B-reps directly and avoids the problem of producing consistent meshing. We observe a similar trend with the ABC dataset. Point cloud methods are unable to discover the topological information necessary to identify rare classes, and suffer from loss of fidelity.

4.2.3 Self-supervised learning

Learning from unlabeled data is important with solid models since real-world labeled datasets are limited, and representation learning by an encoder-decoder scheme is non-trivial due to a lack of B-rep decoders. We leverage contrastive learning [44, 7, 17, 31] (CLR) and propose the following transformations to create positive views for training on B-rep data, each of which enforces a useful prior on the model.

Connected patch Extract a random node and its n-hop neighbors ($n \in \{1, 2\}$). This implies that local patches in a B-rep hint about the global shape.

Drop nodes Randomly delete nodes with uniform probability (0.4) along with attached edges to encourage B-reps with partially similar faces to be clustered together in the latent space.

Drop edges Randomly delete edges with uniform probability (0.4), to encourage B-reps that look similar visually, but have different topology be clustered together.

Our CLR model has three components [7]. Given a B-rep in UV-grid+graph representation G, we uniformly sample two i.i.d. transformations T_1 and T_2 to obtain two positive views $T_1(G)$ and $T_2(G)$. Occasionally (10% of the time), we set T_1 to the identity transformation so that the global shape is available to the neural network to associate with the other partial views. An ablation study for these transformations is provided in the supplementary material.

Our UV-Net encoder extracts the 128D shape embeddings h_i and h_j of positive pairs. A 3-layer non-linear projection head (MLP) with ReLU activations maps these embeddings to 64D latent vectors z_i and z_j. Given a mini-batch of size $N = 256$, we compute $\{z_k \mid k \in 1 \ldots 2N\}$, and bring together the positive pairs while treating the remaining $2(N - 1)$ data as negative examples, with the normalized temperature scaled cross-entropy [7] loss.

We first use the SolidLetters dataset (upper case only since CLR performs per-instance discrimination) to quantitatively understand how our method performs. After training

Table 2: Solid face segmentation. The per-primitive scores refer to per-point scores for point cloud-based models and per-edge scores for MeshCNN. The corresponding per-face scores are computed by voting the predictions from all primitives in a face.

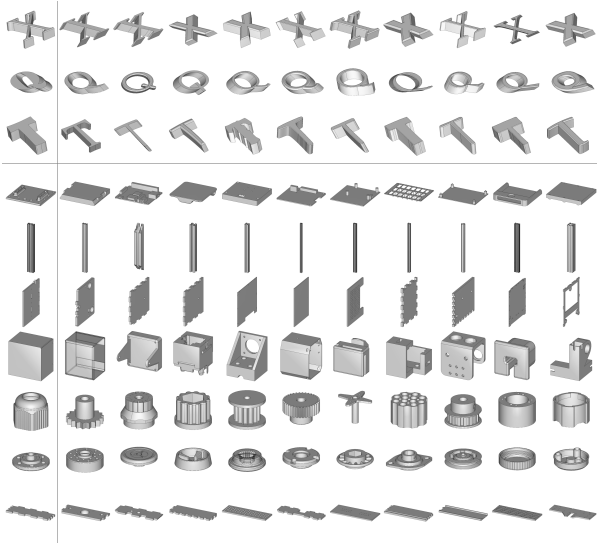| Dataset	Model	Accuracy		Per-class accuracy		Intersection-over-Union		#Param.
		Per-face	Per-prim.	Per-face	Per-prim.	Per-face	Per-prim.	
MFCAD	UV-Net	**99.95 ± 0.02**	-	**99.93 ± 0.20**	-	**99.87 ± 0.03**	-	1.23M
	UV-Net (xyz)	99.83 ± 0.06	-	99.80 ± 0.00	-	99.63 ± 0.06	-	1.23M
	PointNet	32.13 ± 7.92	59.13 ± 7.54	16.20 ± 8.51	15.78 ± 8.17	7.15 ± 5.22	8.27 ± 4.66	0.87M
	DGCNN	82.50 ± 2.46	91.60 ± 2.18	80.43 ± 4.51	78.80 ± 4.57	67.70 ± 4.73	78.67 ± 6.27	0.98M
	GNN	-	-	-	-	93.60 [6]	-	0.53M
	MeshCNN	99.89 ± 0.01	98.52 ± 0.04	99.84 ± 0.03	98.29 ± 0.09	99.70 ± 0.06	95.93 ± 0.05	2.29M
ABC	UV-Net	**88.87 ± 0.70**	-	**56.81 ± 0.93**	-	**50.37 ± 1.11**	-	1.23M
	UV-Net (xyz)	77.33 ± 0.48	-	47.38 ± 0.54	-	38.99 ± 0.42	-	1.23M
	PointNet	40.77 ± 1.79	61.27 ± 0.55	19.87 ± 0.51	25.53 ± 0.32	11.10 ± 0.70	18.47 ± 0.31	0.87M
	DGCNN	54.18 ± 3.19	67.80 ± 0.59	27.30 ± 1.34	34.93 ± 1.52	18.14 ± 1.97	26.26 ± 1.27	0.98M

Figure 4: Self-supervised shape retrieval on SolidLetters and ABC datasets. Column 1: Query, Columns 2–11: Retrieved results sorted left to right by distance in latent space.

Table 3: Quality of self-supervised shape embeddings obtained with our contrastive learning method on SolidLetters.

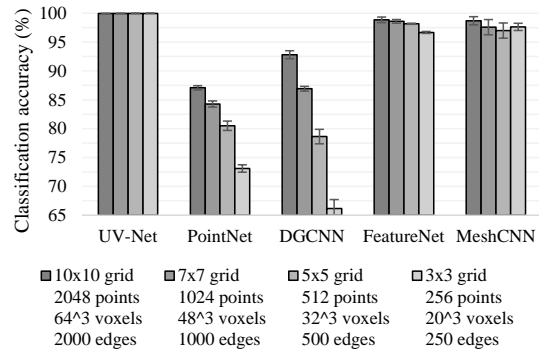| Method	Score (%)
Linear SVM	79.40 ± 0.20
K-means clustering	58.17 ± 0.25

Figure 5: Sensitivity of input representations and methods to sampling resolution for machining feature classification.

the model for 350 epochs, we extract the shape embeddings of the test set and perform k-means clustering to generate 26 clusters. We measure the clustering quality against ground truth clusters (labels) using the adjusted mutual-information metric [37]. We also classify the shape embeddings using a linear Support-Vector Machine (SVM) and compute the classification accuracy. Results in Table 3 show that the shape embeddings obtained with our CLR model is rich with category information even though it is trained without labels.

To perform shape retrieval, we take random shape embeddings from the test set of SolidLetters and ABC (random 20% split) as queries, and compute their k-nearest neighbors

in the UV-Net shape embedding space as shown in Figure 4. The results demonstrate that our CLR approach is viable, and shows high potential to learn from large-scale unlabeled CAD datasets, an unaddressed problem until now.

4.3. Sensitivity to sampling

We now study the effect of the sampling resolution on the accuracy produced by the network. A robust method should degrade gracefully when the sample count is reduced, or leverage other information to produce consistent results. The classification networks are all trained with reduced resolution data and the accuracy is reported in Figure 5. We see that
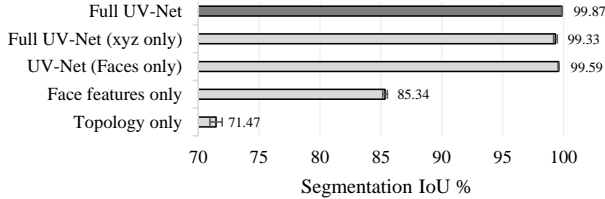
Figure 6: Ablation study with input features and components of UV-Net on the MFCAD segmentation problem.

Table 4: Effect of reparametrizing the SolidLetters classification *test set* on UV-Net.

Convolution	Reparametrized	Test Accuracy
Regular	No	96.74 ± 0.06
	Yes	55.98 ± 2.36
D_2 equivariant	No	96.58 ± 0.01
	Yes	96.59 ± 0.02

our method is robust to sampling resolution in the machining feature detection task. This is because we not only capture the geometry, but also the topological information that can be leveraged for the task. Moreover, every face in the solid is *seen* by UV-Net regardless of its surface area, while other representations suffer from loss of fidelity.

4.4. Feature and architecture ablation

Here we study the importance of the input features, and network components on the MFCAD segmentation problem:

Full UV-Net (xyz only) We remove the normals from the set of input features but use the full architecture.

UV-Net (Face only) This is similar to the full architecture, but without the input curve features and curve CNN ($f_\Theta(h_{uv})$ term in Eq. 1 and entire Eq. 2 are removed).

Face features only We replace the GNN portion of the network with an MLP (similar parameter count as the GNN). Edge features are also removed since they cannot be considered without a message-passing scheme.

Topology only We remove the curve and surface CNNs, and set the edge and node attributes of the graph as noise sampled from a normal distribution, so that the network is forced to solve the task with topology features only.

These networks are trained for 100 epochs on the MFCAD segmentation task and the IoU score is compared against the full model. The benefits of jointly considering the geometric features and topology as proposed is evident from Figure 6, and validates the merits of our approach.

4.5. Invariance to reparametrization

A solid can be altered in subtle ways without changing the 3D appearance by reparametrizing the curve/surface geometry. This can occur when converting models from one format to another (e.g. STEP to SAT), changing the surface type (e.g. plane to spline), or as a result of some high level CAD modeling operation. The UV-grid is to a large extent invariant to common reparametrizations as discussed earlier.

On the other hand, reversing a surface parametrization along the u- or v-axis amounts to flipping the UV-grid about the same axis, while transposing the surface parametrization by exchanging the u- and v-axis is equivalent to rotating and

flipping the UV-grid. Flips about u- and v-axis and rotations by $\{k\frac{\pi}{2} \mid k \in [0, 1, 2, 3]\}$ belong to the Dihedral symmetry group D_2, and regular image convolutions are not invariant to them as we show in Table 4. We see that randomly performing these transformations to surfaces in the test set but not the training set affects the classification accuracy. However, employing D_2 group equivariant convolutions [42] followed by a group pooling layer in the surface CNN (Section 3.1) makes the model resilient to these reparametrizations.

5. Conclusion

We have presented UV-Net, a neural network and representation that can work on B-rep data, and leverage existing image and graph convolutional neural networks. We have shown its benefits and versatility on both supervised and self-supervised tasks spanning five B-rep datasets, outperforming other representations such as point clouds, voxels, and meshes. In addition, we introduced SolidLetters, a new synthetic B-rep dataset with variations in both geometry and topology. We believe our work can unlock data-driven applications in established CAD modeling pipelines, and revitalize research interest in this domain.

Limitations & future work We fixed the sampling step size for each curve or surface regardless of its geometry. Choosing the step using derivatives [47, 20] or learning it in a task-dependent manner could be an interesting extension. While UV-grids are versatile, we did not exploit other information available in the B-rep such as curve and surface types, edge convexity, halfedge ordering, etc. which might prove useful in certain applications. Finally, our UV-grid features are not rotation-invariant. Although we can use local coordinates for each UV-grid [11, 10] or switch to other features like mean-curvature, this may make the network lose sight of the relative orientation of various faces and edges. We leave the detailed study of various invariances to future work. We also believe there is tremendous potential to improve our self-supervised method for transfer learning from large datasets like ABC. Finally, it is worth investigating how ideas from this work can be adapted to other representations like subdivision surfaces, where the limit surface can be parametrized as a regular structure using the faces of the control mesh.

References

[1] Atin Angrish, Benjamin Craver, and Binil Starly. "Fab-Search": A 3D CAD Model-Based Search Engine for Sourcing Manufacturing Services. *Journal of Computing and Information Science in Engineering*, 19(4), 2019. 041006. 5, 12

[2] Silvia Ansaldi, Leila De Floriani, and Bianca Falcidieno. Geometric modeling of solid objects by using a face adjacency graph representation. *ACM SIGGRAPH Computer Graphics*, 19(3):131–139, 1985. 2

[3] Autodesk. Autodesk ShapeManager. https://en.wikipedia.org/wiki/ShapeManager. 6, 14

[4] Bojan Babic, Nenad Nesic, and Zoran Miljkovic. A review of automated feature recognition with rule-based pattern recognition. *Computers in Industry*, 59(4):321–337, 2008. 2

[5] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. 4

[6] Weijuan Cao, Trevor Robinson, Yang Hua, Flavien Boussuge, Andrew R. Colligan, and Wanbin Pan. Graph representation of 3d cad models for machining feature recognition with deep learning. In *Proceedings of the ASME 2020 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, IDETC-CIE. ASME, 2020. 2, 5, 6, 7, 12

[7] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. Big self-supervised models are strong semi-supervised learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 22243–22255. Curran Associates, Inc., 2020. 6

[8] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 42–51, 2020. 2

[9] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5932–5941, 2019. 2

[10] Haowen Deng, Tolga Birdal, and Slobodan Ilic. Ppf-foldnet: Unsupervised learning of rotation invariant 3d local descriptors. In *Computer Vision – ECCV 2018*, pages 620–638, Cham, 2018. Springer International Publishing. 8

[11] Haowen Deng, Tolga Birdal, and Slobodan Ilic. Ppfnet: Global context aware local features for robust 3d point matching. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 195–205, 2018. 8

[12] Jun Gao, Chengcheng Tang, Vignesh Ganapathi-Subramanian, Jiahui Huang, Hao Su, and Leonidas J. Guibas. Deepspline: Data-driven reconstruction of parametric curves and surfaces. *arxiv:1901.03781v1*, 2019. 2

[13] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009. 2

[14] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[15] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. *ACM Transactions on Graphics*, 21(3):355—-361, 2002. 2

[16] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: A network with an edge. *ACM Transactions on Graphics*, 38(4), 2019. 2, 5, 6

[17] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9726–9735, 2020. 6

[18] Yixin Hu, Teseo Schneider, Xifeng Gao, Qingnan Zhou, Alec Jacobson, Denis Zorin, and Daniele Panozzo. Tri-wild: Robust triangulation with curve constraints. *ACM Transactions on Graphics*, 38(4):52:1–52:15, 2019. 2

[19] S. Joshi and T.C. Chang. Graph-based heuristics for recognition of machined features from a 3d solid model. *Computer-Aided Design*, 20(2):58–66, 1988. 2

[20] Taro Kawasaki, Pradeep Kumar Jayaraman, Kentaro Shida, Jianmin Zheng, and Takashi Maekawa. An image processing approach to feature-preserving b-spline surface fairing. *Computer-Aided Design*, 99:1–10, 2018. 2, 3, 8

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 5

[22] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 5, 12

[23] Sang Hun Lee and Kunwoo Lee. Partial entity structure: A compact non-manifold boundary representation based on partial topological entities. In *Proceedings*

of the Sixth ACM Symposium on Solid Modeling and Applications, SMA '01, page 159–170, New York, NY, USA, 2001. Association for Computing Machinery. 1, 3

[24] Raphael G. Lopes, David Ha, Douglas Eck, and Jonathon Shlens. A learned representation for scalable vector graphics. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7929–7938, 2019. 2

[25] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[26] Konstantinos Nezis and George Vosniakos. Recognizing 212d shape features using a neural network and heuristics. *Computer-Aided Design*, 29(7):523–539, 1997. 2

[27] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[28] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer-Verlag, New York, NY, USA, second edition, 1996. 4, 11

[29] S. Prabhakar and M.R. Henderson. Automatic form-feature recognition using neural-network-based techniques on boundary representations of solid models. *Computer-Aided Design*, 24(7):381–393, 1992. 2

[30] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017. 2, 5

[31] Aditya Sanghi. Info3d: Representation learning on 3d objects using mutual information maximization and contrastive learning. In *Computer Vision – ECCV 2020*, pages 626–642. Springer International Publishing, 2020. 6

[32] Thomas W. Sederberg. *"Computer Aided Geometric Design"*. BYU Faculty Publications, 2012. 4

[33] Ayan Sinha, Jing Bai, and Karthik Ramani. Deep learning 3d shape surfaces using geometry images. In *Computer Vision – ECCV 2016*, pages 223–240, Cham, 2016. Springer International Publishing. 2

[34] Dmitriy Smirnov, Mikhail Bessmeltsev, and Justin Solomon. Learning manifold patch-based representations of man-made shapes. *arXiv:1906.12337v3*, 2021. 2

[35] Dmitriy Smirnov, Matthew Fisher, Vladimir G. Kim, Richard Zhang, and Justin Solomon. Deep parametric shape predictions using distance fields. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 558–567, 2020. 2

[36] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 945–953, 2015. 2

[37] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(95):2837–2854, 2010. 7

[38] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (SIGGRAPH)*, 36(4), 2017. 2

[39] Yizhi Wang, Yue Gao, and Zhouhui Lian. Attribute2font: Creating fonts you want from attributes. *ACM Transactions on Graphics*, 39(4), 2020. 2

[40] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 2019. 2, 5

[41] K.J. Weiler. *Topological Structures for Geometric Modeling*. Rensselaer Polytechnic Institute, 1986. 1

[42] Maurice Weiler and Gabriele Cesa. General e(2)-equivariant steerable cnns. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. 8

[43] Ruben Wiersma, Elmar Eisemann, and Klaus Hildebrandt. Cnns on surfaces using rotation-equivariant features. *ACM Transactions on Graphics*, 39(4), 2020. 2

[44] Zhirong Wu, Yuanjun Xiong, X Yu Stella, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018. 6

[45] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. 5

[46] Zhibo Zhang, Prakhar Jaiswal, and Rahul Rai. Featurenet: Machining feature recognition based on 3d convolution neural network. *Computer-Aided Design*, 101:12–22, 2018. 2, 5, 6, 12, 15

[47] Jianmin Zheng and Thomas W. Sederberg. Estimating tessellation parameter intervals for rational curves and surfaces. *ACM Transactions on Graphics*, 19(1):56––77, 2000. 4, 8

[48] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recogni-*

tion (CVPR), pages 1912–1920, 2015. 2

A. Supplementary material

A.1. Sampling error in UV-grids

We quantitatively measure how well the UV-grids approximate the original surface geometry. We chose a random selection of 132,492 models from the ABC dataset, and converted them into our UV-grid representation: curves are represented by grids with 10 points and unit tangents, while surface are represented by 10×10 grids with points and unit surface normals. We then compute four metrics, two related to edge curve approximation and two related to surface approximation:

- Chordal error (curves): The distance between the center of the line joining two points and the ground truth curve evaluated at the average u parameter value of two successive sample points.

- Chordal error (surfaces): The distance between the average of four points defining a patch on a 10×10 point grid and the real surface evaluated at the average (u, v) of the patch. This error metric is considering the point grid as a bi-linear approximation of the surface.

- Bézier approximation error (curves): A cubic Bézier span is constructed from the points and unit tangent vectors following Equation 9.47 in [28]. The Bézier approximation error is then taken as the average distance between the center of the Bézier and the real edge curve evaluated at the average u parameter value of the two sample points used to construct the Bézier span.

- Bézier approximation error (surfaces): A cubic Bézier patch is constructed from the 4 points and unit normals following Equation 9.58 in [28]. The Bézier approximation error is then taken as the average distance between the center of the patch, and the real surface evaluated the central (u, v) parameter value. This error metric is considering the point grid as a cubic Bézier approximation of the surface.

To allow these errors to be compared for solids of different sizes we divide each by then longest length of the bounding box of the entire solid.

As the neural network is passed ordered lists of edge curve points and tangents and an ordered grid of points and normals, the network has sufficient information to understand the curve and surface information as a linear/bilinear interpolation. Chordal errors of **89.19%** surface patches and **93.33%** curves are within 10^{-3} of the longest length of the B-rep's bounding box as shown in Table A.1.

The network also has access to curve tangent and surface normal information. If we assume that the surface normal information can be used by the network then the approximation error is further reduced and we find that the Bézier approximation errors of **96.84%** surface patches and **99.77%** are within 10^{-3} of the longest length of the B-rep's bounding box.

While it is unclear if the network actually uses this information to build an interpolation of the curve/surface geometry, this information is part of the input and is empirically found to help in our ablation studies (see Section 4.4).

A.2. SolidLetters dataset

A publicly available, balanced, and labeled dataset is vital to assist in designing and testing B-rep neural network architectures. To this end, we create "SolidLetters", a new, synthetic, labeled dataset for solid models that includes both geometric and topological variations. It comprises upper and lower case letters in various styles obtained from a collection of 2002 system and Google Fonts. Each data point has three labels: (1) the alphabet, (2) the case (upper or lower), and (3) the name of the font.

Creation We first create the outline of each letter with

Table A.1: Percentage of curves and surfaces with approximation errors exceeding various thresholds computed on random samples from the ABC dataset.

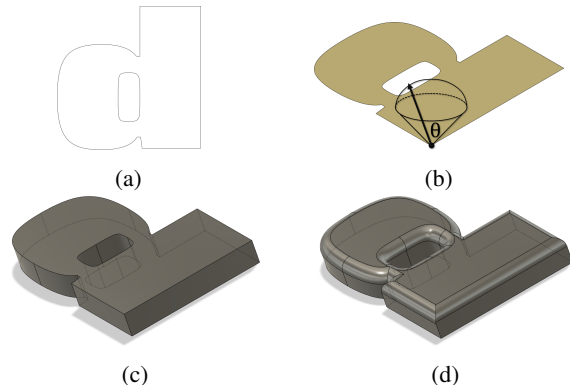| Factor of box size	Surfaces		Curves	
	Bézier	Chordal	Bézier	Chordal
Above 10^{-3}	3.16%	10.81%	0.23%	6.67%
Above 10^{-2}	0.80%	2.65%	0.06%	1.33%
Above 10^{-1}	0.09%	0.10%	0.02%	0.02%

(a)　　　　　　　　　(b)

(c)　　　　　　　　　(d)

Figure A.1: Running example of data generation. (a) 2D wire B-rep going through boundary of the font face. (b) Trimmed planar sheet filling the interior of the boundary. (c) Extrude. (d) Fillet edges of the topmost face (SolidLetters).
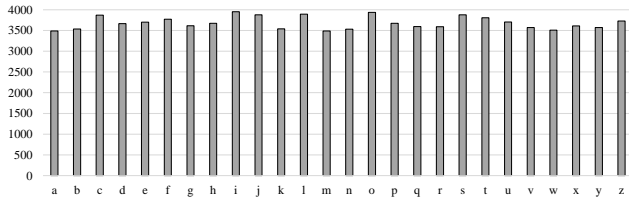
Figure A.2: Per-class distribution of solids in the SolidLetters dataset.

every font (size 10) (Figure A.1a), and fill its interior with a trimmed planar sheet surface, see Figure A.1b. Treating the planar sheet as a profile surface on the XY-plane, we extrude it along a vector **e** pointing upwards, see Figure A.1c. We define this vector such that its head lies at a random point in the spherical cap situated along the z-axis to introduce variance in the extrusion direction. By sampling two random numbers ξ_1 and ξ_2 from a uniform distribution $U(0,1)$, we can define the vector **e** as: $\mathbf{e}_x = \sqrt{1-\mathbf{e}_z^2}\cos(2\pi\xi_2)$, $\mathbf{e}_y = \sqrt{1-\mathbf{e}_z^2}\sin(2\pi\xi_2)$, $\mathbf{e}_z = \xi_1(1-\cos\theta)+\cos\theta$, where θ is the angle subtended by the spherical cap that we set to $45°$. Furthermore, to break the symmetry of the shape across the XY-plane and introduce more complexity in the model, we identify the topmost face in the extruded solid and perform filleting by blending the edges with a constant radius 0.1, see Figure A.1d. This introduces new curved faces in the model along the edges of the topmost face, and changes the topology as well. Filleting is prone to failure when the face has edges that meet at sharp angles or the local thickness is small compared to the filleting radius. Hence, we attempt to fillet three times by successively reducing the filleting radius by 50%, and upon failure leave the extruded solid as such. After removing fonts that are non-English and symbols, we end up with a total of 95,795 data points. There is an average of 33 faces per solid in the dataset. The per-class distribution of data is shown in Figure A.2. We show a visual overview of the entire dataset in Figure A.3.

Data split We partition the dataset into an official 80-20 train-test split based on the complexity of the solids, which can be roughly measured using the number of faces. We place the solids in the datasets into three bins based on the number of faces: $[F_{\min}, F_1), [F_1, F_2), [F_2, F_{\max}]$, where $F_{\min}$ and $F_{\max}$ are the minimum and maximum number of faces in a solid in the entire dataset, respectively. F_1 is defined as $0.15 \times (F_{\max} - F_{\min})$, while F_2 is set to $0.30 \times (F_{\max} - F_{\min})$. The solids in each bin are partitioned randomly into an 80-20 train-test split and finally combined.

A.3. Other datasets

A.3.1 Machining feature

The Machining feature dataset [46] is available at `github.com/madlabub/Machining-feature-dataset`. The original train-test split information was not available, so we created a random 85-15 split within each category, and held out 20% of the training set for validation.

A.3.2 FabWave

The FabWave dataset [1] is available at `dimelab.org/fabwave`. We use the subset of data that the authors call "Standard" which contains mechanical part categories. There are a total of 52 part categories, this is 4 less than what is provided in the dataset because we removed some categories that have very few or no models available in them. There is no official train-test split, so we randomly partitioned the data in a 80-20 ratio within each class. The data distribution is shown in Figure A.4.

A.3.3 MFCAD

The MFCAD dataset [6] is available at `github.com/hducg/MFCAD`. The dataset has 15,488 files (this is 2 more than listed in the paper). The train-validation-test ratio is 60-20-20, and we use the official split shared by the authors which partitions the models while considering the number of labels per solid. Unlike the full set of labels described in the paper, the dataset only has labels on planar faces. This is likely because Cao et al. [6]'s method only supports planar faces. There are a total of 350,295 faces in the dataset classified into 16 segmentation categories. Some visual examples are shown in Figure A.5, and the class distribution in Figure A.6.

A.3.4 ABC

The entire ABC dataset [22] consists of over 1 million CAD assemblies containing over 13 million individual B-rep bodies created by users of the Onshape CAD software. It is available at `deep-geometry.github.io/abc-dataset`. To use the dataset in our experiments we use the following process to remove duplicates and generate segmentation labels.

Duplication removal We remove duplicates from the ABC dataset in four steps. All duplicate removal is performed at the B-rep body level, rather than with assemblies.

1. **Remove small files**: A significant number of models in the dataset are simple primitives that are unsuitable for our experiments. We first remove models with a file size of less than 15kB as a simple but effective proxy for removing simple primitives.

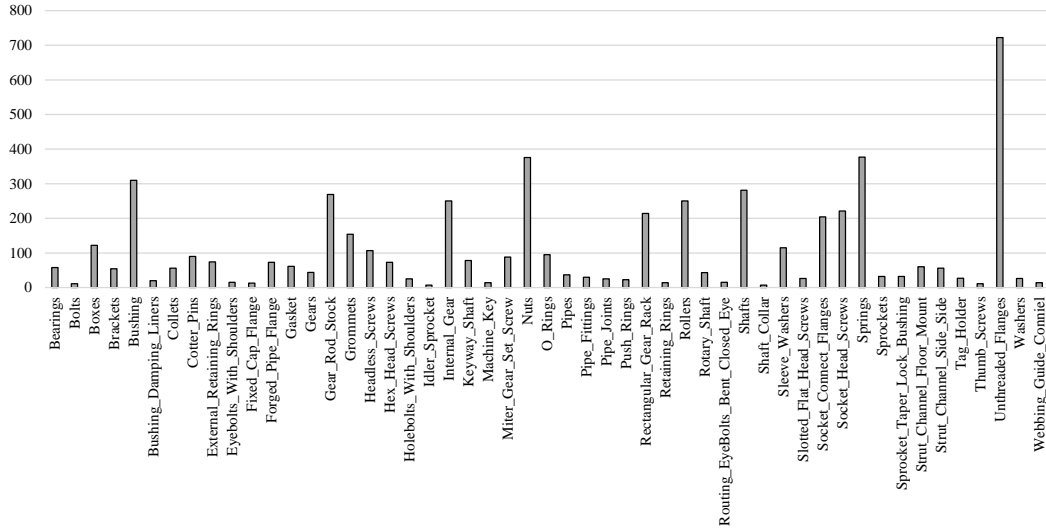Figure A.3: Visual overview of the SolidLetters dataset.

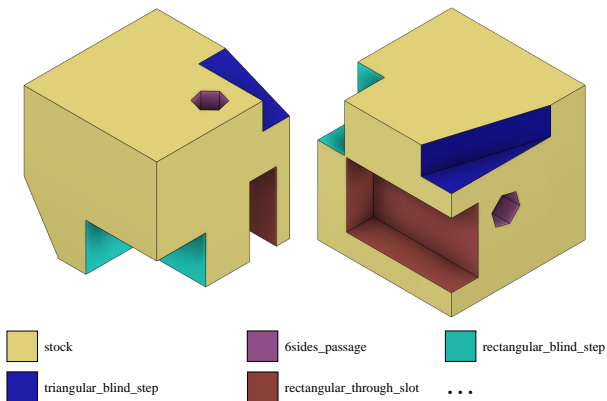Figure A.4: Distribution of categories in the FabWave dataset.

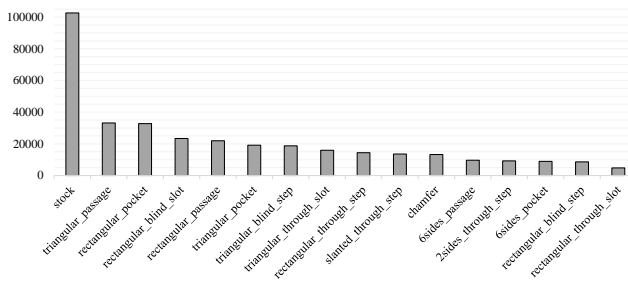Figure A.5: Example 3D models from the MFCAD dataset, colored by segmentation label.

Figure A.6: Distribution of segmentation labels in the MF-CAD dataset.

2. **Remove file duplicates and invalid files**: We next remove exact file duplicates and invalid file type such as `.xmm_txt`.

3. **Remove non-solid and simple solids**: We next remove non-solid models, such as those containing only wires or open solids, as well as simple solids with less than 30 faces.

4. **Remove geometric duplicates**: Finally we remove geometric duplicates by creating and comparing a unique hash string for each model using the number of edges, number of faces, number of shells, number of lumps, area, volume, and moments of inertia. This approach is efficient and invariant to rotation.

From the pool of unique models we choose a random sample of 46k models to use in our experiments.

Segmentation labels Since the ABC dataset is unlabeled, we create our own labels to test UV-Net's segmentation performance on a real-world dataset. We use the Autodesk Shape Manager (ASM) [3] kernel to perform a rule-based feature prediction for each of the faces in the solids. ASM predicts the modeling operation that could have created the face, e.g., chamfer, fillet, extrude and revolve. ASM is unable to identify the modeling operation in some cases, and we ignore such faces during training/testing. Additionally, it also predicts whether the change made by the extrusion was additive or subtractive. We consolidate this information into labels as follows:

• *Chamfer*, *Fillet* and *Revolve* are retained as such. However, we notice that *Chamfer* and *Revolve* are virtually non-existent in our data.
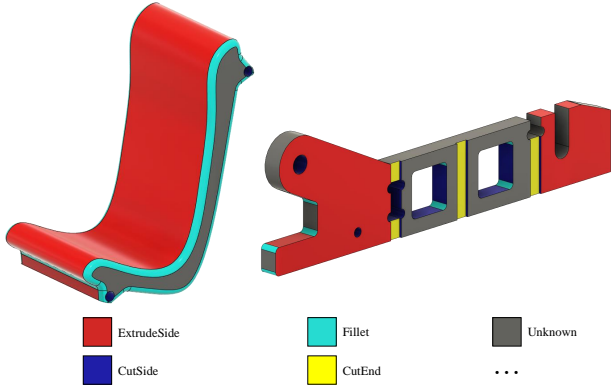
14

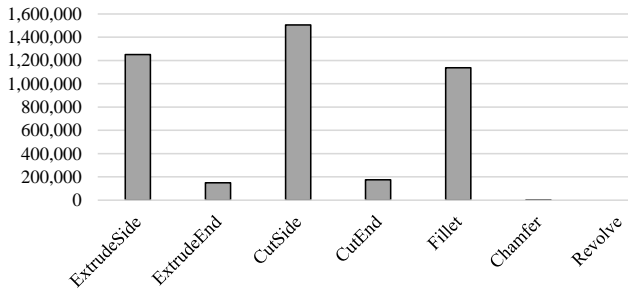Figure A.7: Example 3D models from the ABC dataset, colored by segmentation label.

Figure A.8: Distribution of segmentation labels in the ABC dataset.

- In the case of extrusion, we utilize the extrude direction and the surface normals of a sample of points in the visible region of each face to make a fine-grained categorization.

  - If the normals and extrude direction are aligned, then we set the label as *ExtrudeEnd* if the `change` type is additive, and *CutEnd* if the `change` type is subtractive.

  - If the normals and extrude direction are near perpendicular, then we set the label as *ExtrudeSide* if the `change` type is additive, and *CutSide* if the `change` type is subtractive.

Our subset of the ABC dataset has a total of 4,218,036 faces. Figure A.7 shows some example segmentation labels while the label distribution shown in Figure A.8. The dataset is split into train and test sets randomly in a 80-20 ratio.

A.4. Training details

Our implementation is in PyTorch and we use DGL (`dgl.ai`) for graph operations. All experiments were conducted on NVIDIA GV100, Quadro P6000, or Tesla V100 GPUs. All networks in Section 4 are optimized using the Adam

optimizer with default parameters (learning rate: 0.001, β_1: 0.9, β_2: 0.999).

UV-Net's mini-batches are created by concatenating the nodes and edges of all the graphs in the batch to form a supergraph. For all classification and segmentation experiments, we used a batch size of 128 for UV-Net, PointNet, and FeatureNet. We reduced the batch size to 64 for DGCNN, due to its high memory consumption, and used the default mini-batch size of 16 with MeshCNN. Contrastive learning generally requires a higher batch size since the quality of negative views depend on the data points in the mini-batch, hence, we set it to 256 in this case. DGCNN has a hyperparameter k to define the number of k-nearest neighbors used to build the graph dynamically in each of its layers. We set this to 20 in the classification and segmentation experiments. In the sensitivity to sampling study in Section 4.3, we set k to 10 in the case of 1024 points, and 5 in the case of 512 and 256 points, so that the local neighborhood is well defined.

We adapted the following implementations for our comparisons:

- **PointNet**: we used the PyTorch implementation from the official DGCNN code (`github.com/WangYueFt/dgcnn`) for classification, and an unofficial implementation for segmentation (`github.com/fxia22/pointnet.pytorch`).

- **DGCNN**: we used the official PyTorch implementation available at `github.com/WangYueFt/dgcnn` for classification. Since the segmentation implementation was not available in the official version, we used another implementation from `github.com/AnTao97/dgcnn.pytorch` that is recommended by the authors.

- **FeatureNet**: we implemented this model based on the network architecture provided in the paper [46].

- **MeshCNN**: we used the official implementation from `github.com/ranahanocka/MeshCNN`.

A.5. Additional self-supervised results

A.5.1 Ablation on CLR transformations

We perform an ablation study on the different transformations that we proposed to generate views for contrastive learning. We train our CLR model on the SolidLetters dataset for 100 epochs while removing one transformation at a time. While training for 100 epochs is not sufficient for the network to converge, it gives us a fair understanding of the importance of each transformation. The clustering and linear SVM classification scores are computed as described in Section 4.2.3 and reported in Figure A.9. It is apparent from the results that using all the proposed transformations together is generally beneficial and improves the shape embeddings. It is important to note that the transformations
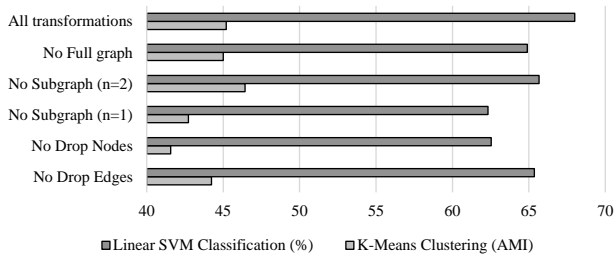
Figure A.9: Ablation on the transformations used in contrastive learning.

may have to be tuned for practical use cases based on the dataset and potential downstream tasks. For example, the right number of hops used to define the subgraphs may vary based on the complexity of the B-reps in the dataset. We did not explore this in our experiments, and directly applied the method that gave best results in the SolidLetters dataset on the ABC dataset.

A.5.2 Shape retrieval

Here we share more qualitative results for shape retrieval on the SolidLetters and ABC datasets with k-nearest search in the latent space generated by our contrastive learning method in Figure A.10 and Figure A.11.
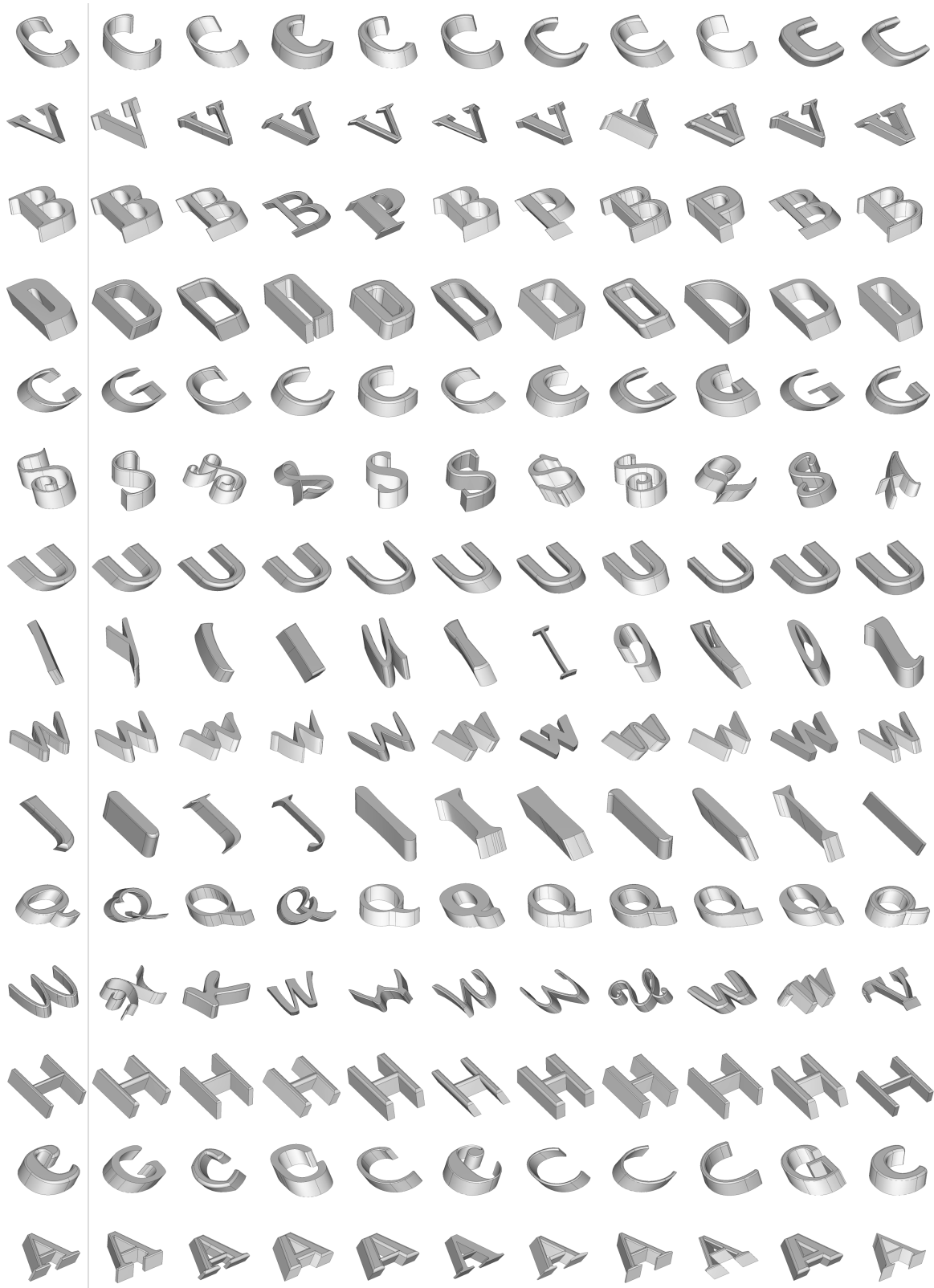
Figure A.10: More self-supervised shape retrieval results on SolidLetters. Column 1: Query, Columns 2–11: Retrieved results sorted left to right by distance in latent space.
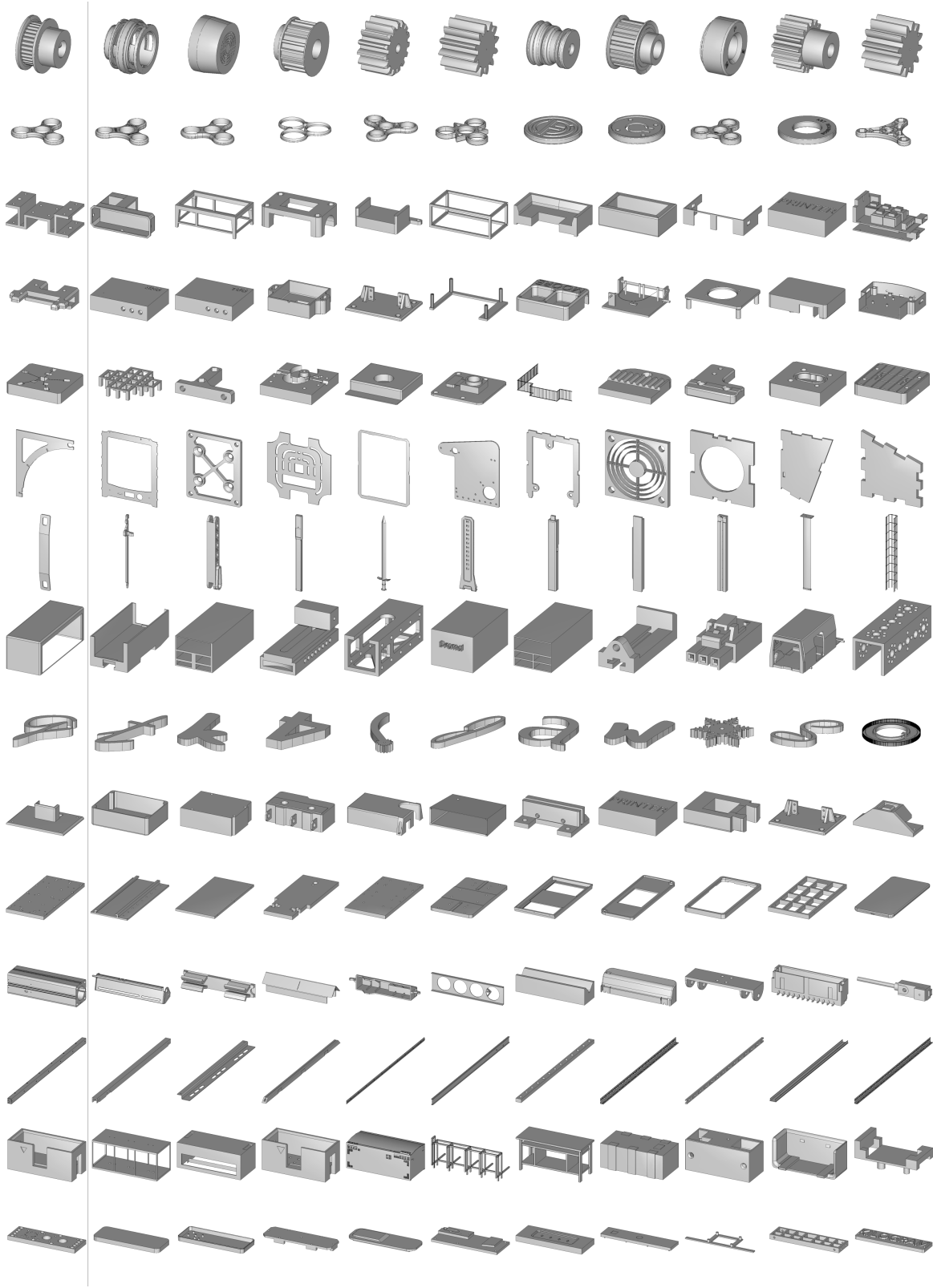
Figure A.11: More self-supervised shape retrieval results on ABC. Column 1: Query, Columns 2–11: Retrieved results sorted left to right by distance in latent space.