A. Introduction to B-Reps

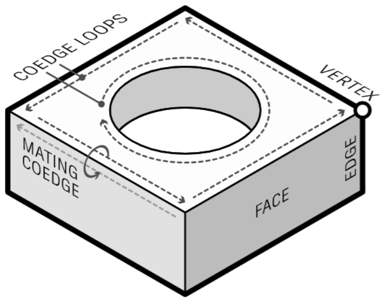


Figure 12: The B-Rep data structure: Faces are defined by parametric surfaces, bounded by loops of trimming curves. Each trimming curve is owned by a topological entity called a coedge, which stores adjacency relationships between faces. Figure from [38].

B-Reps are loosely analogous to 2D Scalable Vector Graphics (SVGs) for 3D. The precise implementation details vary between different CAD softwares, below we describe the general principles relevant to all B-Reps.

As shown in Figure 12, B-Reps are collections of parametric curves and surfaces along with topological information which describes the adjacency relationships between them [38]. They are typically used to describe closed volumes (solids), but can also represent 2D manifolds (sheets) and curve networks (wire bodies). Each face of a B-rep body is defined by a parametric surface which is divided into "visible" and "hidden" regions by a series of trimming loops. The loops comprise an ordered cycle of coedges, which store pointers to "mating" coedges on adjacent faces. The loop ordering and coedge-coedge adjacency information provides a full description of the body's topology, while the parametric curves and surfaces provide the geometric information [26].

B-Reps differ from point clouds and meshes since they are precise representations with continuous smooth surfaces and edge curves — they are not sampled/discrete. Consequently, complex solids may be expressed with low memory requirements without loss of detail [21].

For further information see [38, 26, 21].

B. Few Shot Learning

Figure 14 shows the absolute mean Precision@10 scores over a range of number of positive and negative examples of each of the unseen fonts we tested. These in conjunction with the font shown in Figure 8 (left) are used to calculate the mean gain shown in Figure 8 (right). Examples of each font are given in Figure 13. 1 positive and 0 negative indicates baseline using equal layer weights.

For the most visually distinct fonts (*i.e.* 'Vampiro One' and 'Vast Shadow'), the equal weights baseline is highest. The amount of improvement is dependant on the self-consistency of style within the font and the number of similar fonts in the test set. We observe greater self-consistency within 'Vampiro One' and 'Vast Shadow' while being distinct from the rest of the test set. While the other fonts still show improvement, we expect lower results due to their inconsistency or lack of distinct stylistic features, *i.e.* in 'Stalemate' the 'm' and 's' appear to be stylistically compatible, but the max curvatures of the 'm' are much greater than in the 's' - the style is not obviously the same.

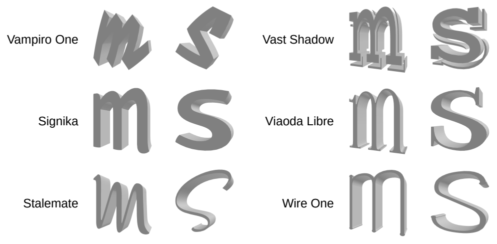


Figure 13

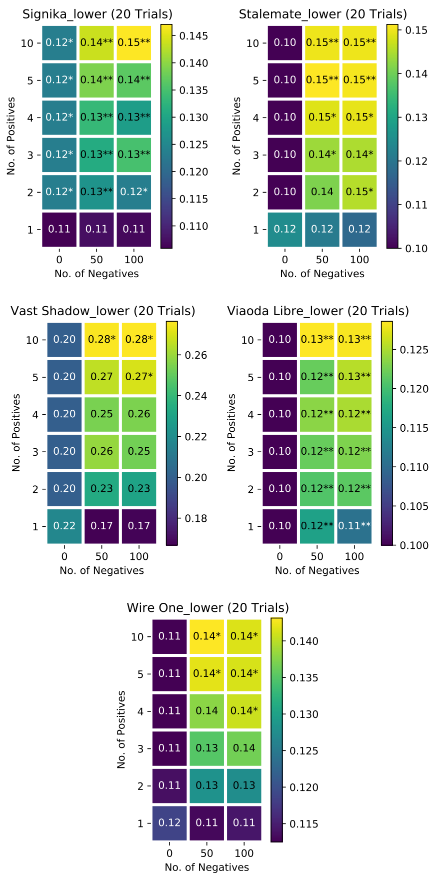


Figure 14

C. Unsupervised Pre-training

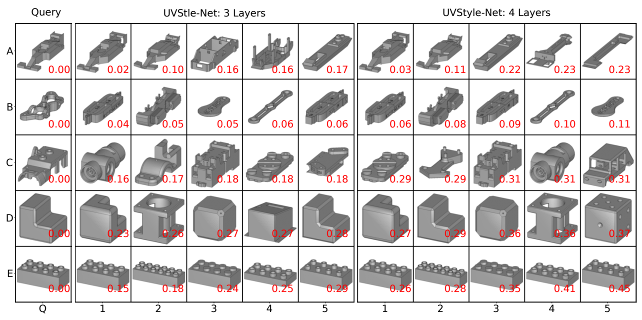


Figure 15: Comparison of top-5 queries with different weights for UVStyle-Net on ABC dataset with unsupervised pre-training. 3 Layers: $\mathbf{w} = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, 0, 0, 0]^\top$, 4 Layers: $\mathbf{w} = [\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 0, 0, 0]^\top$.

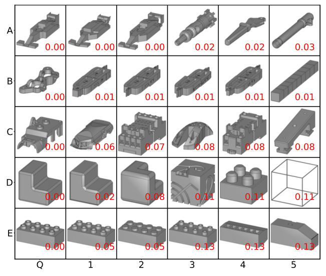


Figure 16: Top-5 query results for ABC dataset from UVStyle-Net with unsupervised pre-training. $\mathbf{w} = [0, 0, 0, 0, 0, 0, 1]^\top$. Weighting the upper layers of the network moves the definition of style closer to content, where the distance measure is more about the general shape and size and global features, and less about the fine details and local features.

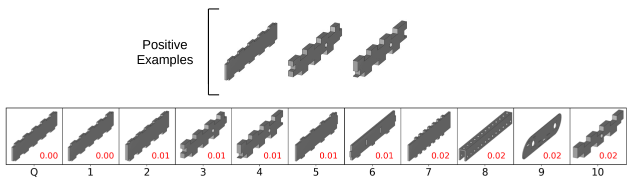


Figure 17: Optimizing $\mathcal{L}_{user}$ with positive examples matching in content results in layer weight distributed over the upper layers. $\mathbf{w}^\star \approx [0, 0, 0, 0, 0, \frac{1}{3}, \frac{2}{3}]^\top$.

D. ABC Style Labels

There is a fundamental lack of publicly available labeled B-Rep data, with no existing B-Rep datasets containing style labels. To enable quantitative evaluation of our method and promote further work in this area we contribute a set of manually assigned style labels for a subset of the ABC solid models. We selected categories with distinct styles while containing diverse content. Examples of each category are shown in Figure 19 and details of the class sizes in Table 3. These labels are available at github.com/AutodeskAILab/UVStyle-Net.

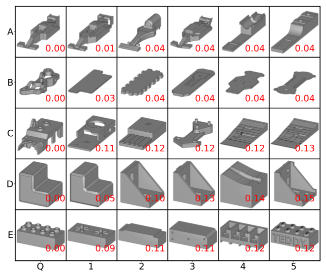


Figure 18: Top-5 queries for PSNet* with cosine distance.

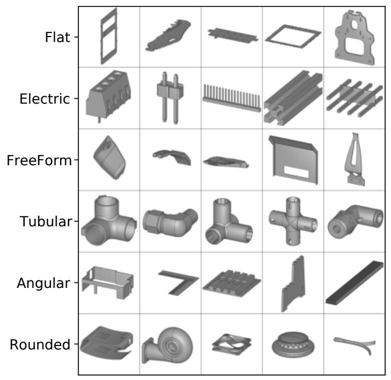


Figure 19: Examples of each ABC style subset classes. Each style is selected to be visually distinct, and while some classes contain the same types of objects, *i.e.*, 'Tubular', the overall shapes (the content) are diverse.

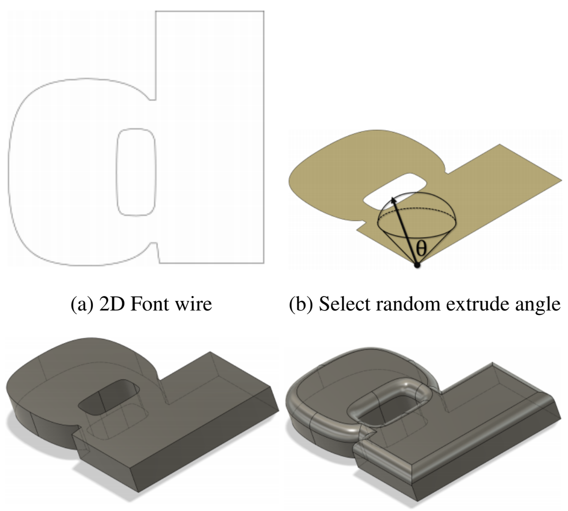
ABC Subset	Examples
Flat/Electric	389/58
Free Form/Pipe	241/24
Angular/Rounded	834/106

Table 3: Manually labeled ABC style subsets.

E. SolidLetters Test Set Generation

For SolidLetters, the training data is generated as per [16] using code and font wires provided by the authors. The key steps are illustrated in Figure 20.

The held-out test set is regenerated to strengthen the associated style labels by removing inconsistent sources of randomness within font classes. The extrusion depth and angle are fixed across all fonts. Filleting size is also fixed,



(a) 2D Font wire (b) Select random extrude angle

(c) Extrude (d) Fillet

Figure 20: Steps for generation of SolidLetters dataset. For test set, extrude angle and fillet amount are fixed. Figure from [16]

and is applied only to fonts where it possible to apply it to all examples of that font. Filleting is not possible for some examples due to the complexity of the solids. If filleting is unsuccessful on any example, all examples of that font are left without fillets.

All SolidLetters data used is freely available at github.com/AutodeskAILab/UVStyle-Net.

F. Model Details

For MeshCNN we use the author's code from `https://github.com/ranahanocka/MeshCNN`, for Pointnet++ we use `https://github.com/erikwijmans/Pointnet2_PyTorch`. All experiments performed on AWS p3.2xlarge.

Table 4 shows details about the model hyper parameters and meta information. For MeshCNN, we remeshed the data to 15000 edges per solid and for Pointnet++ we used their multi-scale grouping (MSG) setup. Other parameters and architecture choices not mentioned here, are set to default. All point clouds are sampled with 1024 points.

For PSNet* we use the Pointnet implementation from `https://github.com/WangYueFt/dgcnn` and extract the Gram matrices from the first 4 layers as detailed in [6]. While PSNet works with geometry and colour, we use only the geometric part in our comparisons.

In Table 5 we compare the computational costs of each encoder.

Model	LR	N	F	BS	Opt
UV-Net	1e-4	BN	7	128	Adam
PSNet*	1e-4	BN	3	128	Adam
Pointnet++	1e-3	BN	6	32	Adam
MeshCNN	2e-4	GN	5	4	Adam

Table 4: Hyper-parameters and meta information about the models for SolidLetters runs. LR denotes learning rate, N type of norm (*i.e.* batch norm or group norm), F input feature dimension, BS batch size and Opt, the type of optimiser used.

Encoder	L	Parameters	Time	Size
UV-Net	7	**645,596**	93min/**88s**	**199 KB**
PSNet*	5	813,914	165min/115s	1.08MB
Pointnet++	22	1,746,420	**43min**/603s	3.32 MB
MeshCNN	5	1,322,982	29hr/38min	305 KB

Table 5: Comparison of 3D encoder methods. L is total number of layers (including features), times given are pre-training/style inference on complete SolidLetters test set. Size is the memory required for a single style embedding (containing one Gram per layer) for a single solid — note this is not dependent on the size of the input solid. For style inference UV-Net is the most compute and memory efficient. MeshCNN suffers from small batch size due to necessarily large meshes, and Pointnet++ suffers from larger Gram matrices.