# Shape Unicode: A Unified Shape Representation

## *Supplementary Material*

Sanjeev Muralikrishnan[1*] Vladimir G. Kim[1] Matthew Fisher[1] Siddhartha Chaudhuri[1,2]
[1] Adobe Research [2] IIT Bombay

This PDF presents:

1. Per-class statistics for the segmentation application described in the main paper.
2. Full architectural details of each encoder and decoder in our framework.
3. A pointer to visualizations of translations, segmentation, correspondences and interpolation.

## 1. Per-Class Segmentation Accuracy.

Segmentation and labeling accuracy of each input representation's Unicode vs its corresponding Solo codes vs ShapePFCN [2] and WU-Net [3], for 16 shape classes in ShapeNetCore [1].

| Category | #train/ #test | #labels | Shape-PFCN | WU-Net | Point Cloud (Joint) | Point Cloud (Solo) | Voxels (Joint) | Voxels (Solo) | Multi-view (Joint) |
|---|---|---|---|---|---|---|---|---|---|
| Airplane | 250/250 | 4 | 88.4 | 90.13 | 85.87 | 86.28 | 87.38 | 86.85 | 86.51 |
| Bag | 38/38 | 2 | 95.5 | 96.02 | 91.48 | 91.41 | 94.52 | 94.43 | 91.95 |
| Bike | 101/101 | 6 | 87.5 | 96.61 | 73.30 | 58.38 | 82.59 | 82.94 | 74.73 |
| Cap | 27/28 | 2 | 92 | 84.77 | 83.37 | 87.58 | 86.48 | 85.27 | 83.99 |
| Car | 250/250 | 4 | 86.6 | 89.92 | 82.84 | 82.40 | 87.03 | 87.03 | 81.55 |
| Chair | 250/250 | 4 | 83.7 | 89.44 | 89.42 | 89.30 | 87.96 | 86.32 | 89.26 |
| Earphone | 34/35 | 3 | 82.9 | 91.82 | 73.28 | 69.40 | 71.22 | 67.32 | 71.31 |
| Guitar | 250/250 | 3 | 89.7 | 78.53 | 94.66 | 94.57 | 95.65 | 95.59 | 94.58 |
| Knife | 196/196 | 2 | 87.1 | 95.98 | 89.43 | 87.70 | 90.85 | 89.74 | 88.90 |
| Lamp | 250/250 | 4 | 78.3 | 90.96 | 84.25 | 82.09 | 78.97 | 63.61 | 84.56 |
| Laptop | 222/222 | 2 | 95.2 | 77.37 | 97.43 | 97.05 | 97.10 | 97.45 | 97.45 |
| Mug | 92/92 | 3 | 98.1 | 99.05 | 98.28 | 98.45 | 98.39 | 98.49 | 98.28 |
| Pistol | 137/138 | 3 | 92.2 | 95.75 | 95.07 | 95.17 | 95.85 | 96.06 | 95.35 |
| Rocket | 33/33 | 3 | 81.5 | 79.94 | 69.12 | 65.28 | 77.51 | 78.37 | 72.27 |
| Skateboard | 76/76 | 3 | 92.5 | 94.66 | 89.10 | 85.37 | 91.34 | 90.19 | 88.20 |
| Table | 250/250 | 3 | 92.5 | 92.91 | 90.83 | 91.21 | 88.74 | 87.45 | 89.83 |
| Category average | | | 89.00 | 90.24 | 86.73 | 85.73 | 88.22 | 86.69 | 86.79 |

Table 1: Dataset statistics and strongly-supervised segmentation and labeling accuracy per category for test shapes in ShapeNetCore, versus ShapePFCN [2] and WU-Net [3], using the splits from [2].

*Corresponding author: samk@adobe.com

## 2. Network Architectures

We now describe the encoder/decoder architectures for each representation, as used to learn Shape Unicode.

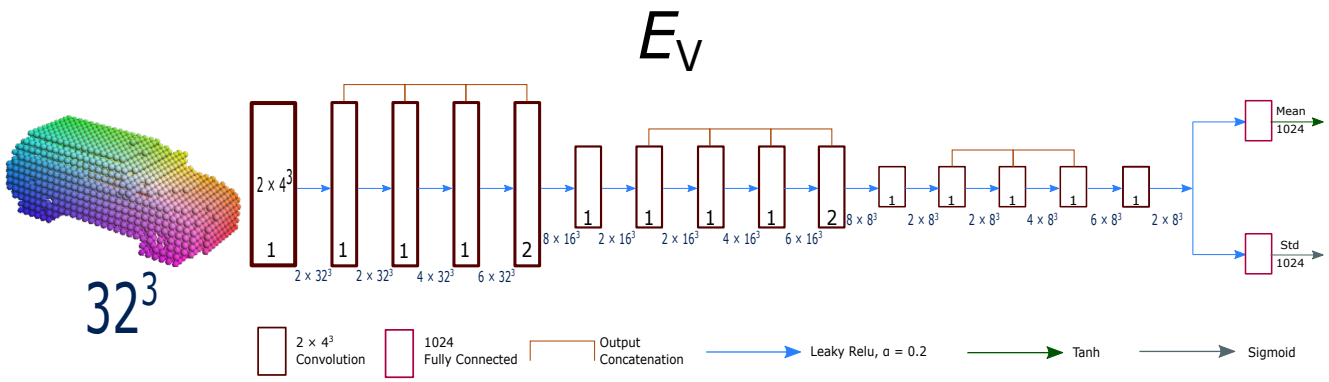


Figure 1: The voxel encoder takes a $32^3$ voxel grid as input and processes it through a series of $4^3$ convolutional layers that each output 2 channels. Each layer's output is concatenated with that of the previous layer, until the next lowres jump. The number inside each block at the bottom is the stride length used at that layer (same for each dimension); Low-res jump is done by using a stride length of 2. The figures at the bottom between layers are the output sizes of the layers. This network outputs a Mean and Standard Deviation using which we subsequently sample the joint code (Unicode).

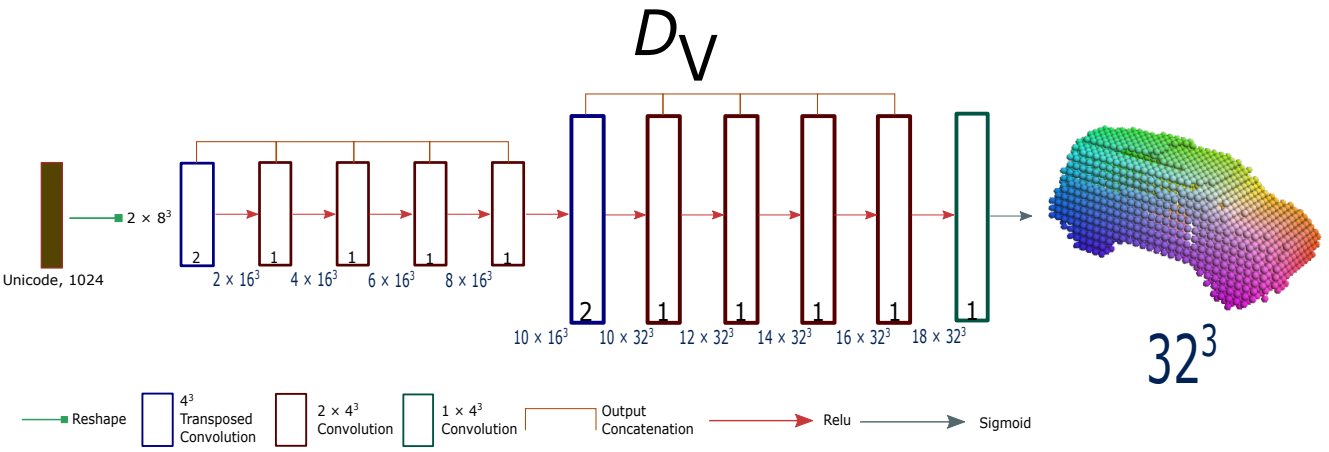


Figure 2: The sampled 1024-dimensional code is rearranged into an $8^3$ grid with 2 channels. This is then processed through a series of transposed and regular convolutional layers. Each layer's output is concatenated with that of the previous layer, until the next high-res jump. Stride lengths are shown as in Figure 1, and stride 2 transposed convolutions are used for the high-res jump. This network outputs the reconstructed $32^3$ voxel grid.

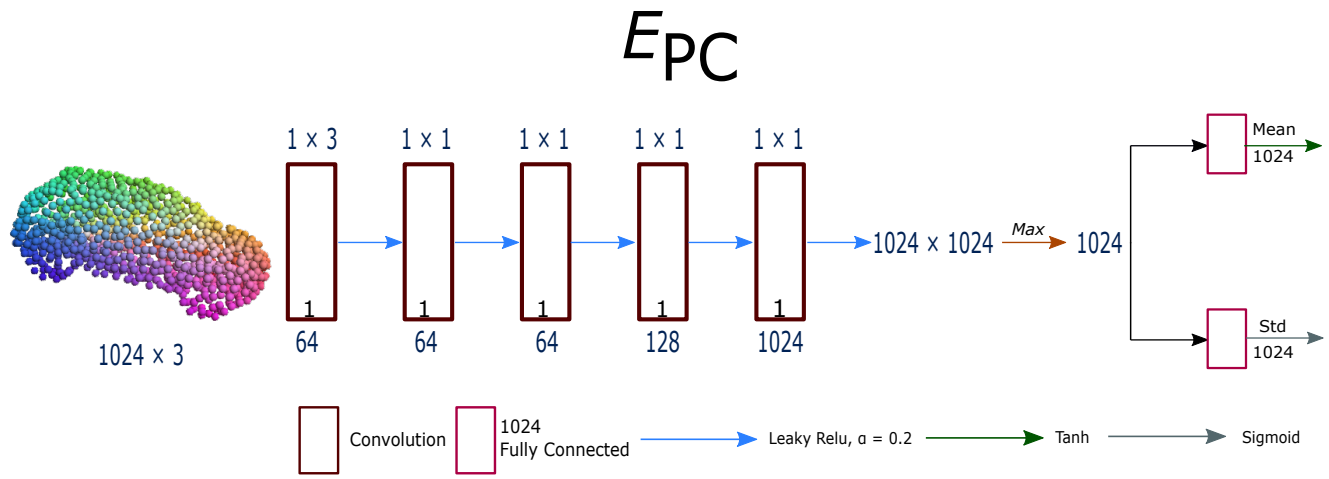


Figure 3: The point cloud encoder takes a $1024 \times 3$ point cloud (1024 points with 3 coordinates each) as input, which is processed through a series of convolutional layers. The hyper-parameters of each layer are shown above: the number above each layer is the filter size, the number at the bottom *inside* each layer is the stride length, and the one *outside* is the number of output channels. This network outputs a 1024-D feature for each point, after which a global max-pooling is performed to obtain a 1024-D intermediate shape code. Then, via fully connected layers, this network outputs a Mean and Standard Deviation using which the joint code can be sampled.

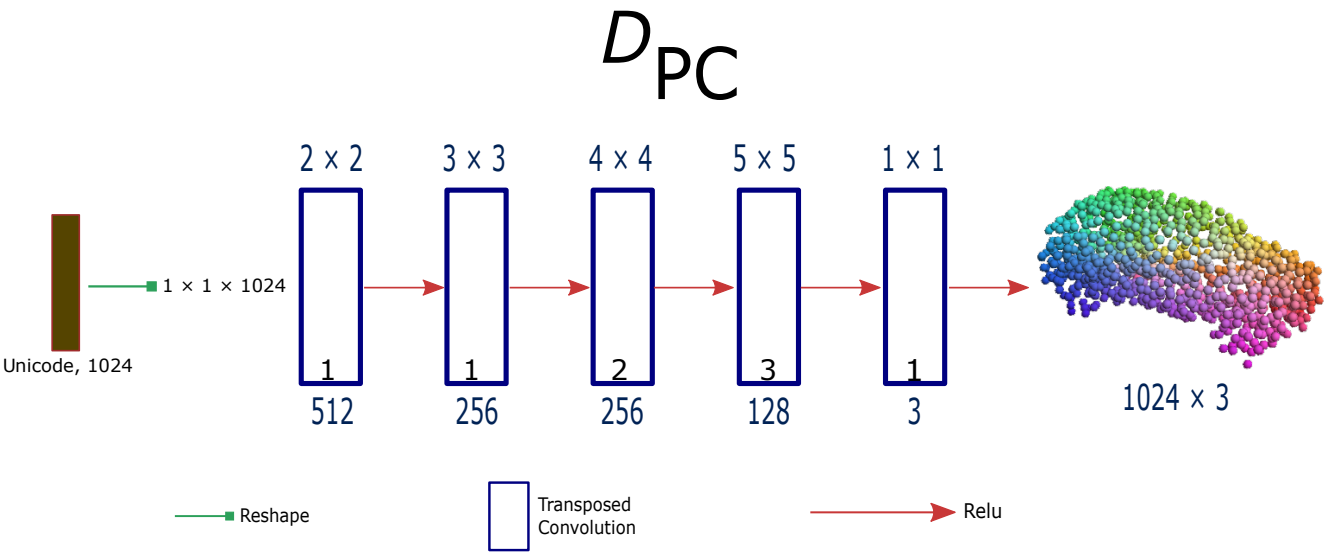


Figure 4: The sampled 1024-dimensional code is sent through a series of transposed convolutions with hyper-parameters indicated above, as in Figure 3. This decoder outputs the $1024 \times 3$ reconstructed point cloud.

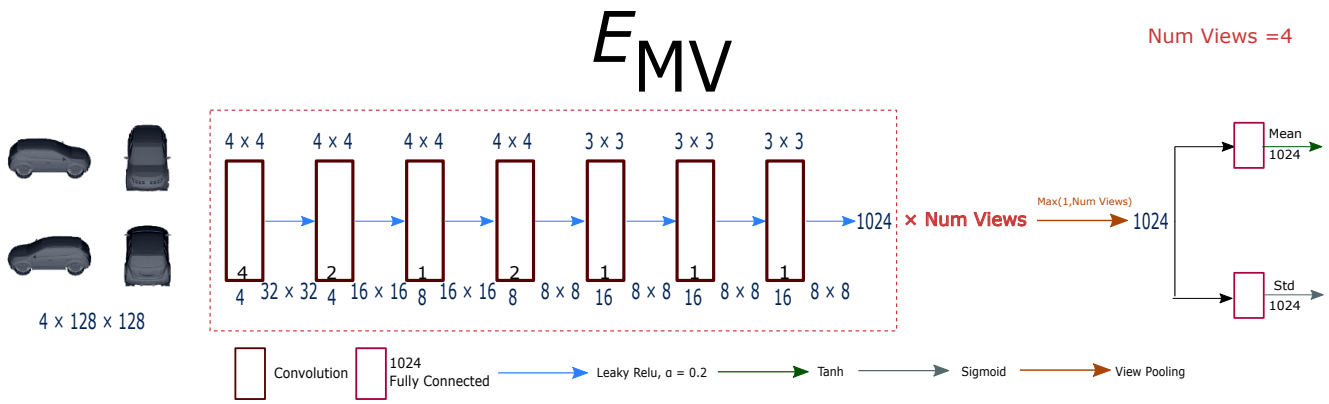


Figure 5: The multi-view encoder processes 4 input view images, with each processed through a different view encoder. Each view encoder has the same architecture and hyperparameters, as illustrated above, without any weights-sharing. The hyperparameters of each layer are shown above: the size on top of each layer is the filter size, the number at the bottom *inside* is the stride length and the one *outside* is the number of output channels. The size between layers is the image size after the previous layer is applied. Each such view network outputs a 1024-D code, after which we apply max view pooling to obtain a single 1024-D intermediate code. Then, via fully connected layers, we output a Mean and Standard Deviation using which the joint code can be sampled.

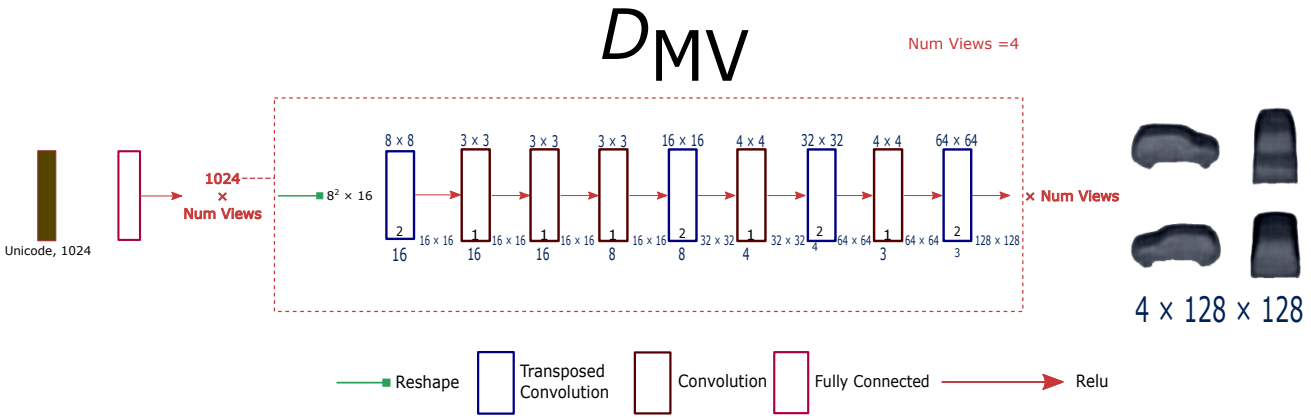


Figure 6: The sampled joint code is first sent through a fully connected layer that outputs 4 1024-D codes, one for each view. Each view code is then sent through its own view decoder, with each decoder having the same architecture and hyper-parameters, but without any weight-sharing. Each view decoder takes in the 1024-D code of the corresponding view and rearranges it into an $8^2$ image with 2 channels. This is then processed through a series of transposed and regular convolutions. The hyperparameters of each layer are illustrated in Figure 5. Each such view decoder outputs a $128 \times 128$ reconstructed view image.

## 3. Supplementary Visualizations

We provide visualizations of shape translations, segmentations, correspondences and interpolated generations in additional supplementary material on the project website for this paper (space constraints prevent us uploading the full directory of images to the CVPR repository).

## References


[1] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An information-rich 3D model repository. *CoRR*, abs/1512.03012, 2015. 1

[2] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri. 3D shape segmentation with projective convolutional networks. In *CVPR*, 2017. 1

[3] S. Muralikrishnan, V. G. Kim, and S. Chaudhuri. Tags2Parts: Discovering semantic regions from shape tags. In *CVPR*, 2018. 1