SVGformer: Representation Learning for Continuous Vector Graphics using Transformers

Defu Cao[1], Zhaowen Wang[2], Jose Echevarria[2], Yan Liu[1]
[1]University of Southern California
[2]Adobe Research, Inc.
{defucao, yanliu.cs}@usc.edu, {zhawang, echevarr}@adobe.com

A. SVG data format

Scalable Vector Graphics (SVG) used in this paper is an XML-based format for vector graphics, which contains a set of paths to build hierarchical and complex graphs. Each path consists of several commands with different types. We follow the command type of DeepSVG [1]. To feed the SVG input into the neural networks, we focus on 3 specific types of commands with corresponding arguments:

- **MoveTo** (M). M means the start of a new path in our structure and the arguments (x_2, y_2) indicate the path stats at point (x_2, y_2).
- **Line** (L). L means the computer needs to draw a visible line from the endpoint (x_1, y_1) of the last command to the point (x_2, y_2), where the point (x_2, y_2) is the arguments of this command.
- **Cubic Bézier** (C). C means the computer needs to draw a visible cubic Bézier curve from the endpoint of the last command to the point (x_2, y_2) with control point (q_{x_1}, q_{y_1}) and (q_{x_2}, q_{y_2}), where the point (q_{x_1}, q_{y_1}), (q_{x_2}, q_{y_2}), (x_2, y_2) is the arguments of this command.

These three basic types of commands can be combined sequentially to build complex geometric objects, including lines, circles, polygons, and splines without losing precision in an efficient way. With all the arguments, we can build the tensor with shape $n \times 8$ as the input continuous value of the given SVG, where n is the number of commands and the eight positions are $x_1, y_1, q_{x_1}, q_{y_1}, q_{x_2}, q_{y_2}, x_2, y_2$ from left to right. In addition, each SVG has a corresponding commands type matrix as the label of the training task, which can be used to plot the input SVG and the predicted label can be used to plot the reconstructed SVG.

B. Datasets details

Fonts Dataset: We directly extract the fonts SVG from ttf files provided by Google Fonts at https://github.com/google/material-design-icons and filter the files with commands over 60. We split the dataset to train/validation/test parts with the rate of 80%/10%10% with 162,115 training samples. Please refer to their official homepage for more detailed descriptions and settings: https://fonts.google.com/.

Icons Dataset: We use the SVG-Icons8 dataset from [1] which consists of 100,000 icons in 56 different categories, considering the consistency and diversity for training samples. Besides, we filtered the icons with commands over 50 or with paths over 8. It can be downloaded from https://github.com/alexandre01/deepsvg. We can recover the SVG format using the library provided in the above link and build our input according to the data format mentioned in Appendix A. The division is the same as the previous DeepSVG.

C. Baselines

LayoutTransformer [2]: LayoutTransformer introduces a transformer-based auto-regressive model that leverages self-attention to learn the contextual relationships between different vector graphic elements. It can handle the inherent symmetries

with different attributes of vector graphics by modeling them separately. In this work, we briefly modify the framework and add a commands type classification loss to make LayoutTransformer fit for the SVG representation learning task. However, it only accepts the discretization input so that we discrete the continuous input in the same way as DeepSVG. The source code can be found at https://github.com/kampta/DeepLayout/tree/main/layout_transformer. We use the default configuration given by the repository with 50 epochs and the data split is in the same way as SVGformer.

DeepSVG [1]: DeepSVG is a hierarchical generative network specifically for vector graphics animation. It can jointly utilize the information from high-level shapes and low-level commands in a hierarchical structure. DeepSVG is a strong baseline and the source code is available at https://github.com/alexandre01/deepsvg. Instead of directly using the pre-trained model, we trained the DeepSVG on the Fonts dataset and the filtered Icons dataset separately for a fair comparison. Specifically, we first build a font meta file to guide the DeepSVG to find all the samples by their library and then train the model on the dataset with discrete and normalized values.

D. Model detail

We conduct our experiments using four NVIDIA GeForce GTX 2080 GPUs. The inputs are normalized with a view box (0, 0, 24, 24). Note that, the reconstruction results are transformed back to the original scale, and metrics are calculated on the original data. Specifically, we trained SVGformer for 50 epochs with a learning rate of 0.0001, 0.05 dropout, and an early stop strategy if the validation loss did not change within a certain range in 3 epochs. The batch size is 32 for both Fonts and Icons datasets. For the embedding layer, we used a look-up table with a fixed size of 47 for mapping the segment information into the embedding vector, where we can map each segment label to a unique 512-dimensional vector. Thus, the relationship between the same label commands can be enhanced from the geometric analysis. Note that, our MAT implementation sometimes fails on un-closed SVG path, and we need to pad such unprocessed path with the same segment label (0) and mask the geometric relationship of such cases.

The raw input size of the encoder and decoder before embedding layer are both $n \times 8$ and the feature vector size after the embedding layer is $n \times 512$. The number of encoder layers is 2 and the decoder layer number is 1. In each encoder layer, we use an 8-head self-attention to calculate the sparse semantics attention score and inject the GCN into the attention score at the very beginning of the encoder layer, where we can take advantage of the pure geometric information before the data is involved in the data-driven stage. The sparse semantics attention takes the top 50% of Q order by the M from equation (5). The GCN layer has an input size of 512 and an output size of 512 to directly do the operation on semantics attention score. The decoder layer is stacked with our spare attention mechanism and the fully self-attention mechanism with 8 attention heads and followed by the GELU activation function. Besides, the fully-connected layer is needed to yield the final logits to the loss function. In addition, we directly fine-tune SVGformer on the Icons dataset using the pre-trained Fonts model as our transformer-based model needs more data to learn the representation which fits for multiple sketches and diverse styles instead of just applicable for one particular kind of SVG.

E. More results

E.1. Reconstruction Results

Figure 1 and Figure 2 show some new results on the reconstruction task, which illustrates that SVGformer can provide notably higher reconstruction quality compared with previous baselines. Note that, LayoutTransformer can handle some of the easy SVGs better than DeepSVG, while it fails with some mess lines and random endpoints when facing complex shapes. The precise reconstruction results indicate that SVGformer can fully utilize the sparse self-attention mechanism to learn the semantic information of the SVGs and utilize the geometric information from SVG input to make sure the sketch is aligned with the original input.

E.2. Retrieval Results

Figure 3 and Figure 4 further demonstrate how the representation of SVG can provide meaningful information by performing the retrieval task on both Fonts dataset and Icons dataset. Note that, the retrieval task is a fully unsupervised task which needs the representation that can be identified with the detailed features. We can still observe significantly better retrieval capability than previous baselines. As more and more SVG images become available in the real-world, the powerful retrieval capability will provide designers with the assurance and insights by collecting images with the same styles.

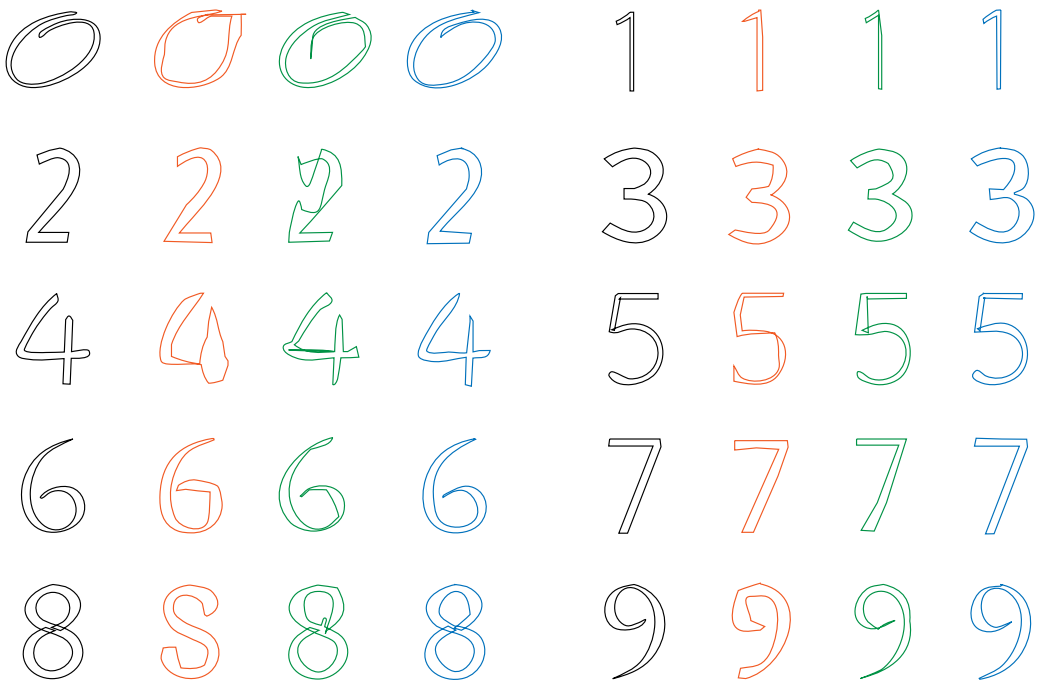

Figure 1. Reconstruction results of Fonts. From left to right, the original SVG (black), DeepSVG (orange), LayoutTransformer (green) and our SVGformer (blue), respectively.

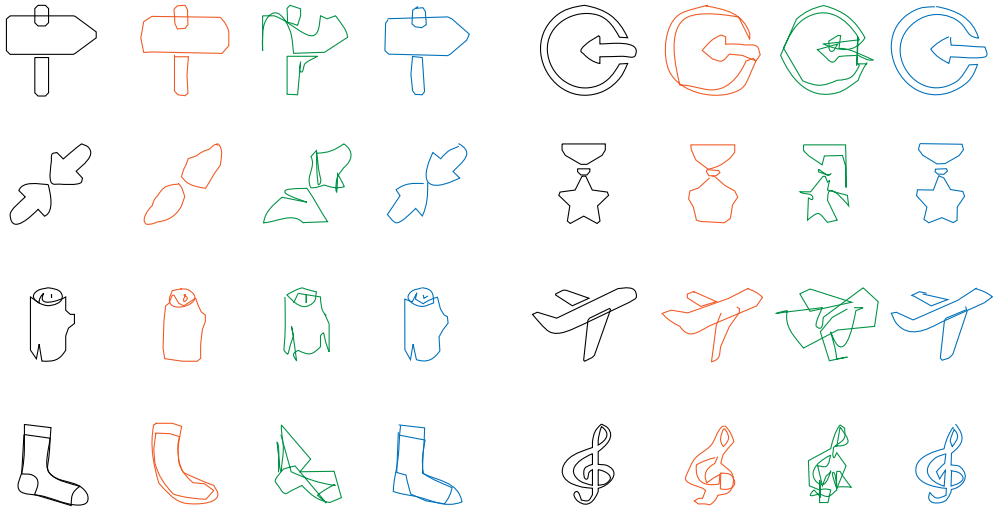

Figure 2. Reconstruction results of Icons. From left to right, the original SVG (black), DeepSVG (orange), LayoutTransformer (green) and our SVGformer (blue), respectively.

E.3. Interpolation Results

Given the strong representation ability of reconstruction, we further ask if our latent representation is smooth and linear enough to investigate the interpolation task. In this task, we provide several groups of two similar SVGs to perform the internal process of how the in-between SVGs generate and how the shape morphs. Note that we manually choose the similar SVGs so that we can verify if the middle results in reasonable or not which can provide more insights for the real-world design application. We first show the interpolation ability of single cases in the Icons dataset in Figure 5. Given the more smooth and more precise reconstruction results, SVGformer can smoothly interpolate between the two different paths while the DeepSVG

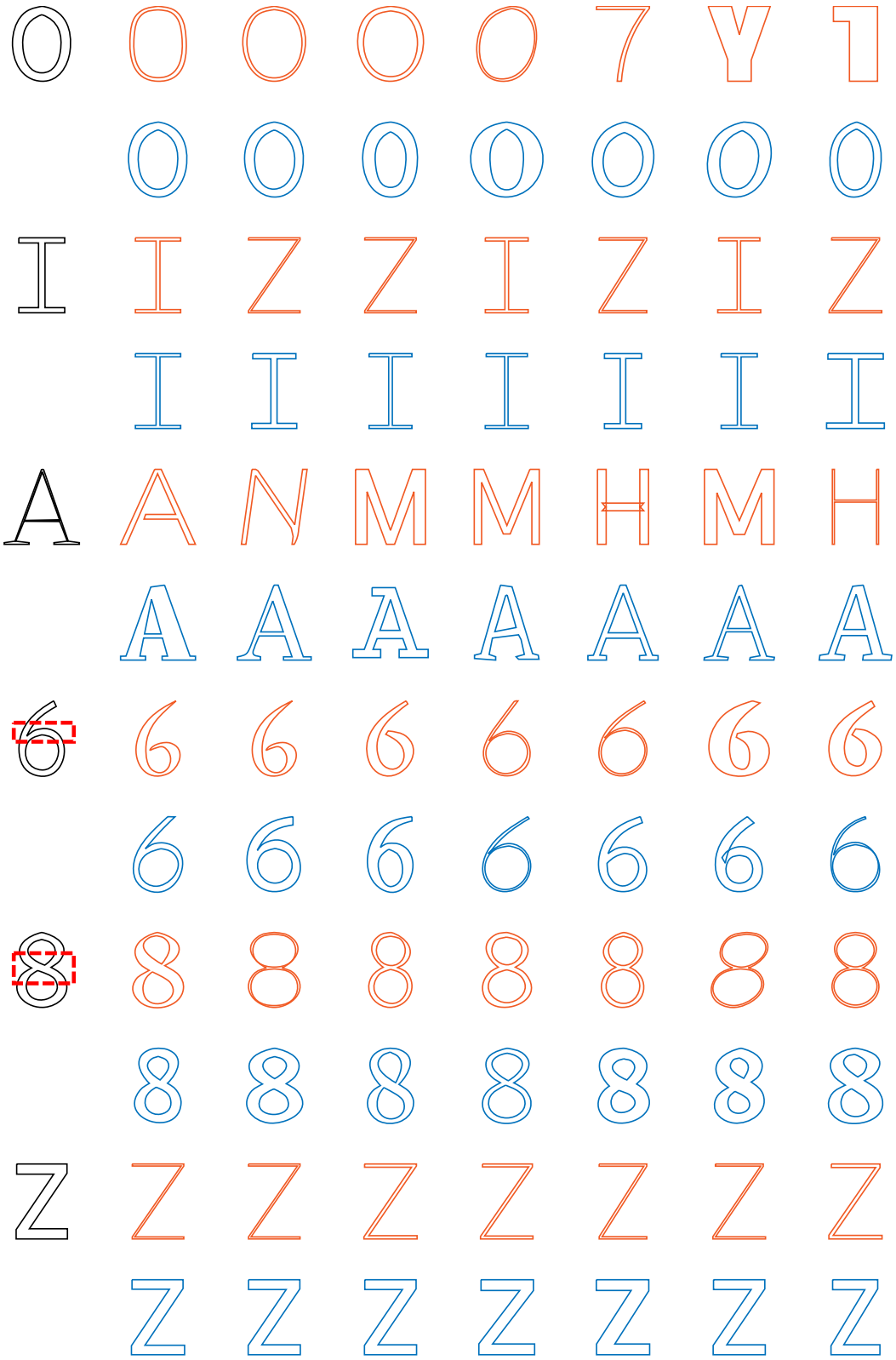

Figure 3. Retrieval results of Fonts. Results for DeepSVG are shown in orange, and results for SVGformer in blue. For the retrieval results in the same scope, we find that our model can distinguish the detailed features. For example, the retrieval results of DeepSVG on "6" always return the cases with one path, while the query has two single paths. The key feature needs to be distinguished is shown in in first red dotted rectangle. In addition, The retrieval of "8" requires the model has the ability to discern between the circle and drop shapes (shown in second red dotted rectangle), where SVGformer accomplishes a better job than DeepSVG.

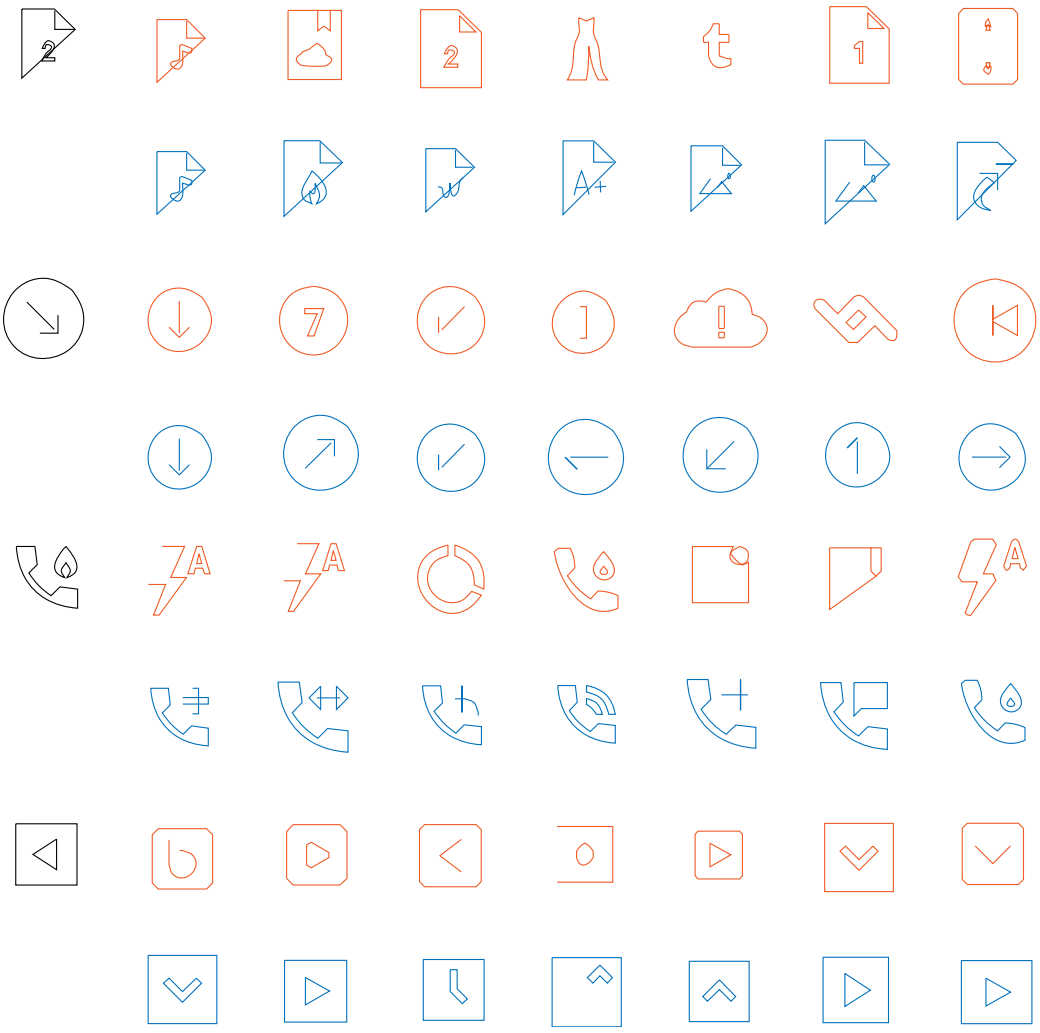

Figure 4. Retrieval results of Icons. Results for DeepSVG are shown in orange, and results for SVGformer in blue.

cannot perform well in such structures without hierarchical components. For the more challenging SVGs, we conduct two different ways to use the SVGformer of interpolation task. The first way (second line in each examples in Figure 6) is directly utilizing the decoder to generate the middle representation's shape at one time. The second way (third line in each example) is that we can split the two target SVGs into several related parts and then use the decoder to finish the interpolation task separately, which is inspired by the hierarchical structure of DeepSVG. As shown in Figure 6, the hierarchical SVG can yield reasonable results while SVGformer without utilizing the hierarchical decoders can be failed at some inner stage. However, we find that the hierarchical structure is also useful for SVGformer in complex SVGs, which can be a direction of future work.

F. Case Study

As discussed in the limitation section, we found that a good representation alone is not enough for some complex interpolation tasks. In other words, complete and smooth interpolation requires a good representation and a strong decoder that understands the representation to work together. To demonstrate that our representation is good and reasonable, we conducted an extra analysis in terms of the representation similarity of simple single paths with cosine similarity scores, for example, squares and rectangles, in Figure 7. Note that, the encoder needs to consider the permutation invariance as the order of input data sequence should not influence the representation in a significant way. However, several related works [2–4] suffer the limitation that the order of the different elements does have a significant impact on the performance. As shown in Figure 7, the similarity results show that our model can handle such invariance with a much more robust representation than DeepSVG on

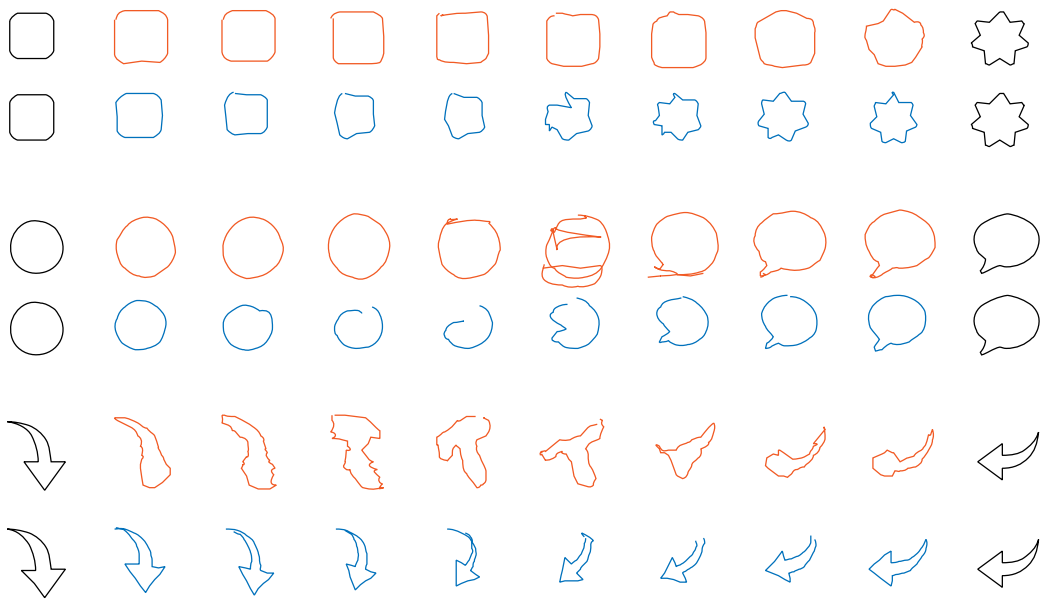

Figure 5. Interpolation results of Icons with a single path. Results for DeepSVG are shown in orange, and results for SVGformer in blue. For the single path, SVGformer can handle it in a better way compared with DeepSVG's mess line. For example, we can find some arrows with different orientations in the third sample's SVGformer results, while the DeepSVG cannot yield ideal interpolation results.

similar shape pairs. Moreover, we plot more cases in Figure 8 where SVGformer sometimes cannot interpolate in a smooth way. In those failure cases, we found that SVGformer can still match similar parts from different SVGs and get more accurate reconstruction results compared with DeepSVG. Therefore, we conjecture that our decoder may not be strong enough for smooth interpolation of complex shapes, which motivates us to improve our decoder in future research.

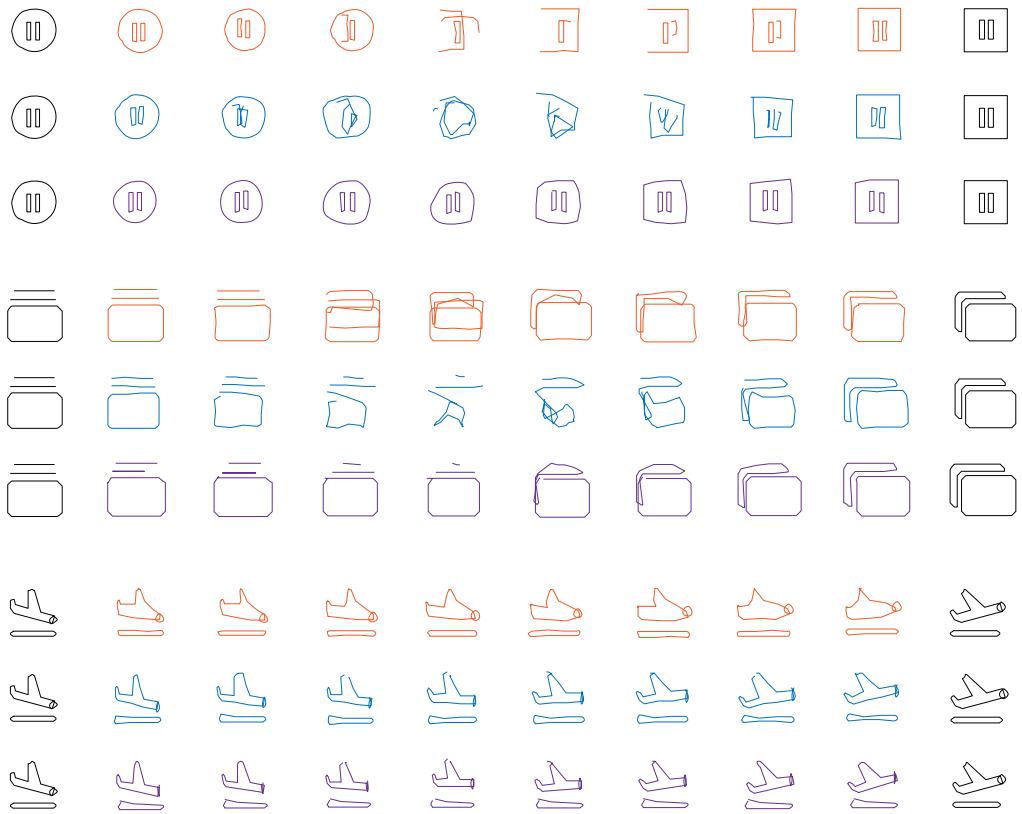

Figure 6. Interpolation results of Icons with multiple paths. For multiple paths, the hierarchical structure plays an important role in enhancing the smoothness of the interpolation process. The first row in orange for each case is the results of DeepSVG, and the rest are the results for different SVGformer with different strategies. Compared with directly using the decoder on all paths of one SVG (second row in blue for each case), SVGformer can yield more smooth and more reasonable in-between results after incorporating the hierarchical operation (third row in violet for each case).

	Deep-SVG	SVG-former		Deep-SVG	SVG-former
	0.80	0.95		0.93	0.97
	0.82	0.95		0.49	0.95
	0.72	0.93		0.96	0.98
	0.82	0.97		0.74	0.94

Figure 7. Cosine similarity of learned representation on pairs of similar icons. It can be seen that both position and orientation can have an influence on the hidden representation. However, our method is less sensitive to the transformation in position, orientation, and size, which shows that SVGformer can produce a more robust latent representation.

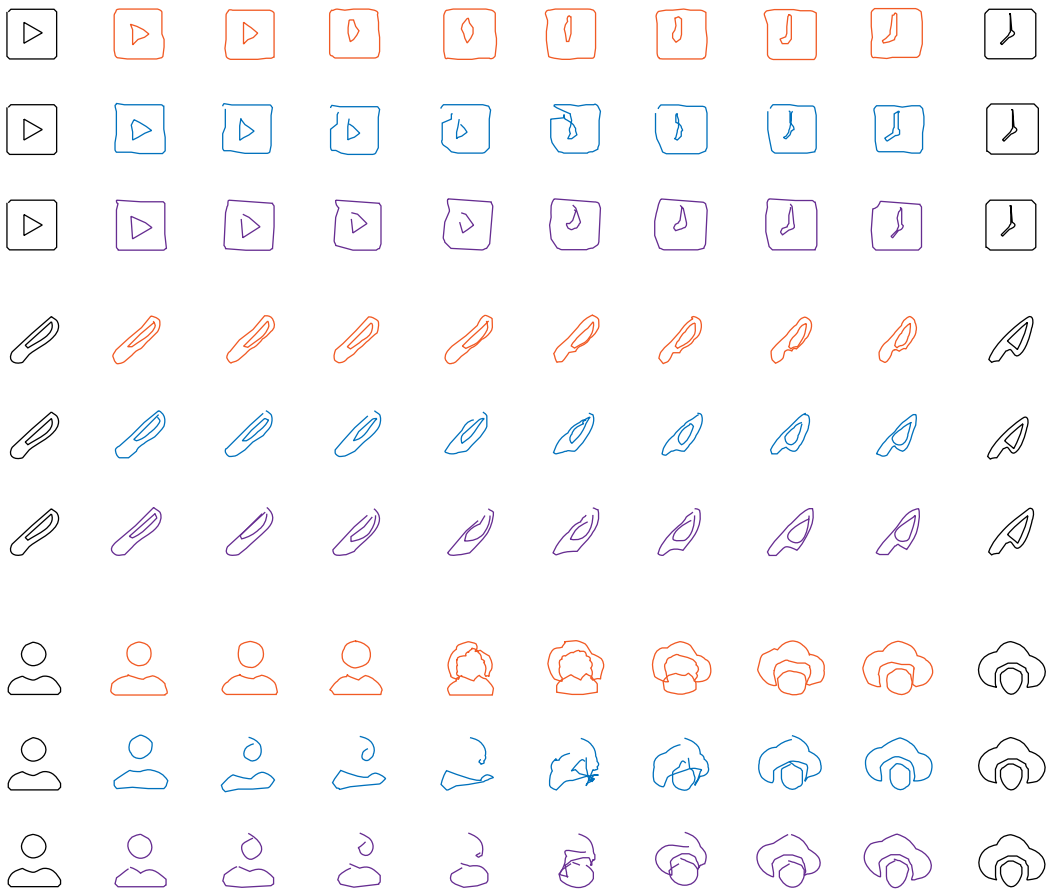

Figure 8. Case study on interpolation results of Icons with multiple paths. Results for DeepSVG are shown in orange, results for SVGformer with one encoder in blue, and results for SVGformer with the hierarchical encoders in violet. We find that SVGformer sometimes can return non-smooth interpolation with broken lines or irregular shapes. In the first example, the outline squares of two anchor shapes are pretty similar, with similarity scores of 0.90 on DeepSVG and 0.91 on SVGformer. However, the interpolation results of SVGformer are not ideal compared with DeepSVG, even though the results have a higher similarity with the anchors. As we already verify the representation ability of SVGformer's encoder part, we conjecture that the decoder module is as important as the encoder module in order to get smooth and faithful interpolation results.

G. Chamfer distance (CD) metric

We introduce how we can calculate the CD metric for the reconstruction taskin this section.Specifically, we first collect the point cloud $\Phi_i = \bigcup_j^{N_C} \phi_{i,j}$ for path P_i by sampling n_p points on each SVG command C_i^j and calculate the point value ϕ_i: $\phi_{i,j} = \{\mathbf{r}_{i,j}\,(k/n_p)\}_{k=0}^{n_p-1}$ if $c_i^j \in \{\mathrm{L}, \mathrm{C}\}$ and $\phi_{i,j} = \varnothing$ otherwise, where $\mathbf{r}_{i,j}(t)(0 \le t \le 1)$ is parametric representation of the curve C_i^j. The point cloud of the reconstruction path $\hat{P}_i$ can be calculated similarly and denoted as $\hat{\Phi}_i$. Then we can calculate the chamfer distance for the two corresponding paths by:

$$
\begin{aligned}
&d_{\mathrm{CD}}\left(\Phi_i, \hat{\Phi}_i\right) = \\
&\frac{1}{|\Phi_i|} \sum_{\mathbf{x}\in\Phi_i} \min_{\mathbf{y}\in\hat{\Phi}_i} \|\mathbf{x}-\mathbf{y}\|_2^2 + \frac{1}{\left|\hat{\Phi}_i\right|} \sum_{\mathbf{y}\in\hat{\Phi}_i} \min_{\mathbf{x}\in\Phi_i} \|\mathbf{x}-\mathbf{y}\|_2^2.
\end{aligned} \tag{1}
$$

The average Chamfer distance is $\mathcal{L}_{\mathrm{cfr}} = \frac{1}{N_P} \sum_i^{N_P} d_{\mathrm{CD}}(\Phi_i, \hat{\Phi}_i)$.

References

[1] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. Deepsvg: A hierarchical generative network for vector graphics animation. *Advances in Neural Information Processing Systems*, 33:16351–16361, 2020. 1, 2

[2] Kamal Gupta, Justin Lazarow, Alessandro Achille, Larry S Davis, Vijay Mahadevan, and Abhinav Shrivastava. Layouttransformer: Layout generation and completion with self-attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1004–1014, 2021. 1, 5

[3] Rundi Wu, Yixin Zhuang, Kai Xu, Hao Zhang, and Baoquan Chen. Pq-net: A generative part seq2seq network for 3d shapes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 5

[4] Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3d-prnn: Generating shape primitives with recurrent neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017. 5