DeepSVG Recon – AI2612 Final Project

Letian Yang
moekid101@sjtu.edu.cn

Jiude Wei
wjd_kznwl@sjtu.edu.cn

Yuying Zhang
shjtzyy01@sjtu.edu.cn

Abstract

Scalable vector graphics (SVG) is one of the most crucial graphic formats in AIGC considering its scalability, editability, interactivity and lossless compression. We have reproduced Deepsvg with default fonts dataset. Furthermore, we have collected and refined 14000 SVGs representing weapons, weather and clothes, and attempted to modify the neural network and corresponding configs to fit our dataset.

1 Introduction

AI-driven content generation has become more mature and has been accepted and enrolled as an indispensable factor in more creative content generation occasions, especially in artificial voice synthesis, text generation and graphic generation. The scalable vector graphics (SVG) are especially commonly used in graphic generation tasks considering that 1) SVGs are resolution-independent and can be scaled to all sizes without losing resolution quality. 2) SVGs are based on XML format and hence easy to be interacted with AI programs. 3) The lines, fillings and colors can be controlled precisely by accessing the content of SVGs.

In this paper, we have trained a nerual network that produces SVGs representing Arabian numbers. We are also inspired to collect and refine more SVG training data that represent weapons, weather and clothes. Then we uniform the formats of the collected SVGs and modify the neural network and the configurations that preprocesses the training data so that the preprocessed training data enable our neural network with recoginizing and producing the colors of lines in SVGs correctly.

2 Related Work

The guide to this project recommends us to use DeepSVG as the model to predict SVG images.

DeepSVG is a Hierarchical Transformer-based generative model that is capable of encoding and predicting the generate commands that constitute SVG images. DeepSVG effectively disentangles high-level shapes from the level commands that encodes the shape. The encoder of DeepSVG has capacity of exploiting the permutation invariance of input SVG by seperately encoding its every shape and then producing the latent vector. The decoder of DeepSVG mirrors the encoding approach by predicting a set of shape representations along with their associated attributes and then combines the shapes into the output SVG image.

DeepSVG project also proposed *SVG-fonts* and *SVG-Icons8* dataset. *SVG-fonts* constitutes of nearly 200k SVG icons representing Arabian numbers from 0 to 9 for font generation. *SVG-Icons8* is composed of SVG icons with consistency and diversity representing meaningful and generalizable real-world items.

3 Data Collection & Refinement

The early-term task requires us to collect SVG images illustrating weapons, weather and clothes on websites by Python crawlers. We built a crawler program based on the well-constructed crawler

template given by T.A and executed it on several websites that store SVG icons. We then refined the collected SVG images manually and selected out nearly 20k colorful and representative SVG images.

3.1 Data Collection

Crawler. The crawler is separated into 3 stages. 1) We enumerate web pages that contains Urls pointing to web pages of SVG images, with essential key words including theme, coloring and figure sizes. 2) We extract the source paths of SVG images from the Urls, then combine every source path with its responding Url into pairs for saving SVG images later. 3) For each source path-Url pair, we extract the SVG command from the html text from the source path.

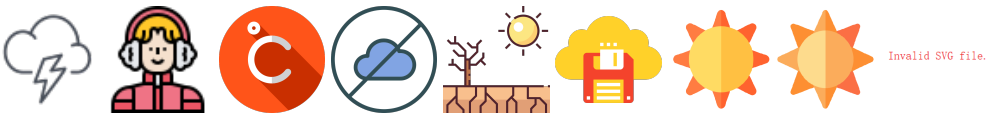
Source. We collected SVG images on icon666.com due to several reasons. 1) This website has collected SVG images from outer websites, therefore it contains huge amount of colorful and representative SVG images of decent themes. 2) SVG commands are clearly extracted from Urls of this website. 3) A general header is available on this website, making it easier to adapting crawler for collecting SVG images.

3.2 Refinement

Format. We observed that the collected SVG images have difference in the format of their commands and hence some are even incorrectly rendered. Therefore we unified the format of the collected SVG images. More specifically, we 1) removed redundant headers and comments in the SVG commands; 2) replaced keyword *viewbox* with *Viewbox* for valid and correct rendering.

Representation. Despite our effort on modifying the commands of SVG images, some images are still invalidly rendered. So we removed all invalidly rendered SVG images. We also believe colors are essential for images being representative as well as images with representation in a narrow sense strengthens the connection between the theme and image itself. Therefore we filtered our collected data under following regulation. 1) Monochrome images are should be removed. 2) Images that do not directly represent the required theme should be removed. 3) Images representing required themes with letters or words should be removed. 4) Images with negative meanings should be removed. 5) Images representing multiple parallel items in concept level should be removed. 6) Images representing wrong themes should be removed. We also removed similar SVG images that share the same label to simplify our dataset. Fig. 1 gives examples of retained images with label clothes, weather and weapon. We also present Fig. 2, showing examples of removed images with label weather.

Figure 1: Examples of retained SVG images with label clothes, weather and weapon.



(1) (2) (3) (4) (5) (6) (7) (8)

Figure 2: Examples of removed SVG images. (1) is monochrome. (2) shows a boy in heavy clothes to represent cold weather, which requires association. (3) uses letter symbol of celcius. (4) contains a banning sign. (5) represents drought with multiple items: land, dying tree and sun. (6) actually represents cloud storage rather than weather. (7) shows two similar-looking sun with slight difference in color, then only one of them is retained. (8) is a sample of invalidly rendered SVG image.

The number of SVG images we collected by crawler and we refined are shown in Tab. 1.

Label	#Crawled Images	#Refined Images	Retainment Rate
Clothes	9230	4158	0.450
Weather	10523	832	0.079
Weapon	13965	4793	0.343

Table 1: Number of images we crawled and selected.

4 Experiments

We have conducted experiment on *SVG-fonts* dataset presented by DeepSVG as well as our dataset containing SVG images with theme **weapons**, **weather** and **clothes**. Furthermore, the original function to process and unify the format of SVG images may leads to lack of element or invalid rendering of SVG image, therefore we improved the function to enable it to correctly transforming the formats and every modified SVG image has similar appearance with its pre-modified version.

4.1 Fonts Generation *(Task 1)*

In this section, we have trained DeepSVG with *SVG-fonts* dataset with configuration file *hierarchical_ordered_fonts* to generate Arabian numbers and English letters (both upper case and lower case). As shown in Fig. 3, we have trained a model that generates synthetic fonts with desirable performance.

However, from the given example of Arabian number **0**, upper-case letter **G** and lower-case letter **a** in Fig. 3, we have already identified the drawbacks of framing the SVG generation task as an NLP-style sequence prediction challenge. The generation model is **not equipped with spatial awareness and spatial modelling capabilities**, as evidenced by the crossing of the two edges in the (5)-th and (6)-th subfigure.

(1) (2) (3) (4) (5) (6) (7) (8) (9)

Figure 3: Examples of generated gonts

4.2 Icon Generation on Custom Dataset *(Task 2)*

From the previous subsection we have had some idea on how to manipulate the utilize DeepSVG for model training. In this subsection we will transplant these insights into training an SVG icon generation model on our own dataset (selected from what is provided and our crawled icons from icon666.com as is described in Section. 3).

This subsection fulfills the specifications outlined in *Task 2*.

4.2.1 Substituting Datasets

We begin with understanding the *hierarchical_ordered_fonts* configuration file and the *fonts_meta* meta-data file. The most crucial column in the CSV file for aiding the model in distinguishing labels is **uni**, corresponding to the **filter_uni** attribute of *Config* class. Therefore, we can simply assume the three classes (weapons, weathers, clothes) are three types of letters in some certain "font". Consequently, after preprocessing each class, we merge the three meta-data files into one single CSV with column **uni** allowing for three possible values: 48 for clothing, 49 for weather and 50 for weapon. Correspondingly, the **filter_uni** attribute is set to [48, 49, 50].

Then the problem arises that most of our data (9783 SVG files) "vanishes" after the preprocessing stage, leaving only 1677 (less than 20%) samples available. It turns out that in *deepsvg.svg_dataset* module, *SVGDataset* class filters the dataset through restricting the maximum number of groups and maximum length of each group (**which implies that our data is too complex for *deepsvg***).

After hours of debugging, we have finally discovered that we should change not only the *max_num_groups* and *max_seq_len* in class *Config*, but we should concurrently adjust the corresponding attributes in *ModelConfig* class. We suffered from the asynchronous initialization of *max_total_len* and *num_groups_proposal* in *_DefaultConfig* class of *deepsvg.model.config* module. Moreover, the errors induced by consistency issues between these parameters remain dormant until the training phase. Hence, we can only deduce the cause of the errors from the Tensor shape inputed to *SVGTransformer* class at training stage.

Having completed all debugging, we set parameters *max_num_groups*$= 15$ and *max_seq_len*$= 40$, thereby retaining 6477 of the original 9783 SVG files. Notably, this is a **tradeoff between data complexity and amount of training data**, filtering out the samples that are too complex while maintaining most of the dataset available.

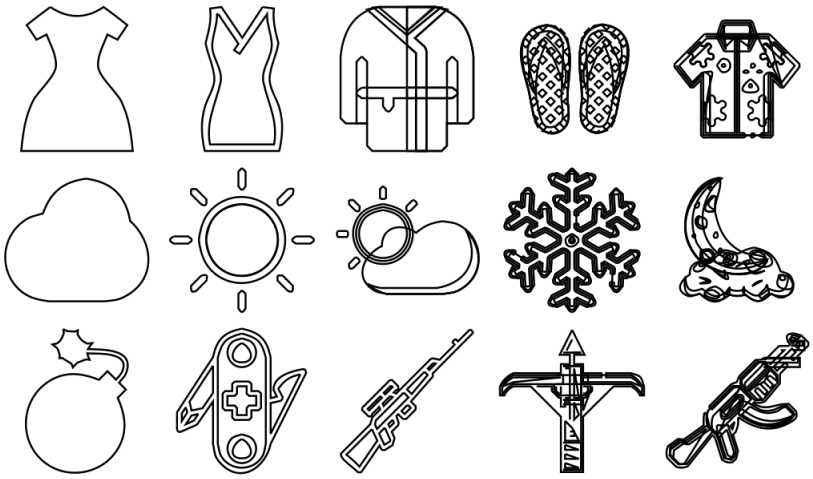


Figure 4: Samples from our dataset. Each row demonstrates one category (clothes, weather and weapon). Within each row the samples are arranged in the order of ease (easiest in the left while hardest in the right).

Fig. 4 demonstrates 15 samples from all 3 different categories in our training dataset. It is pretty straightforward that there exists **significant intra-class variations**, **super-hard SVG samples** and **complicated textures**, all adding to the difficulty of training an SVG icon generator. (Notably, all samples here are preprocessed to be binary-colored and there is no filling.)

4.2.2 Sampling and Interpolation

With the problems arising from complex datasets solved, we performed training process keeping all other parameters unchanged. However, as is shown in Fig. 5, the model is entirely incapable of extracting unique features for each SVG category, while the only noteworthy achievement is that the model generates distinct outputs for different classes.

(1) (2) (3) (4) (5) (6) (7) (8) (9)

Figure 5: Examples of generated icons, sampled with temperature $t = 0.3$. The leftmost 3 images are sampled from category clothes, the middle 3 from weathers while the rightmost 3 from weapons.

Considering the exceptionally unsatisfactory performance of the trained model, we gave up interpolating between generated icon pairs, instead we consider interpolation only between real icon pairs. There being little noteworthy programming and debugging issues, we directly present our interpolation results.

In Fig. 6, we see how interpolation works between icon pairs. We see from the interpolation results that the model encoder-decoder framework could barely capture the shape information of SVG images for extremely simple samples. With this observation, we conclude that simply eliminating hard samples from our dataset does not help improve icon generation. Instead, **the generation fails largely due to intra-class variations**. The reasoning are stated as follows.

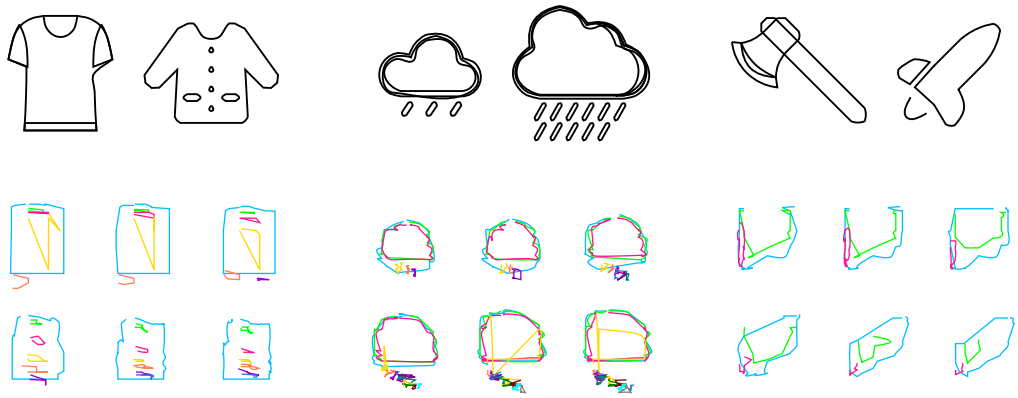


Figure 6: Example result of interpolation for each category. The images in the first line demonstrate original SVG images, while the second line shows the interpolation images.

In the interpolation process, we are sampling $\boldsymbol{z} = \alpha \boldsymbol{z}_0 + (1 - \alpha)\boldsymbol{z}_1$ for every encoded pair $(\boldsymbol{z}_0, \boldsymbol{z}_1)$. When α approaches either 0 or 1, we witness (relatively) meaningful outputs (take the rainy weather interpolation as example). However, when α is not close enough to 0 or 1, the "mixture" of two SVG samples are close to meaningless unstructured scribbling, especially when the selected $\boldsymbol{z}_0$ and $\boldsymbol{z}_1$ differs much. If we want to generate a meaningful SVG image, we want the generation decoder to start from a latent vector $\boldsymbol{z}'$ close to some certain sample in the dataset, through which a meaningful generation becomes possible.

However, in our training process, there are only < 10000 samples with the dimension of latent vector $\boldsymbol{z}$ being 256. Therefore, it is very obvious that **the dataset samples are very sparse in the latent space**. In this case, intra-class variations will likely draw the latent vector to some point that isn't represented by any training sample, composing meaningless SVG image.

By contrast, the fonts generation succeed due to very small intra-class variations, where a certain unicode character (which clearly won't differ much across fonts) is encoded to a whole latent space, allowing the latent vector $\boldsymbol{z}$ to take arbitrary random values while ensuring a relatively fixed shape context. Consequently, in order to improve the performance, we have to **conduct a more nuanced segmentation of the dataset labels**.

5 Further Discussion

Having analyzed the performance of *DeepSVG* from a dataset-centric perspective, we reevaluate it from the model-centric standpoint. Essentially speaking, we raise the following question: **is *DeepSVG* a thoroughly effective solution?**

The authors have modelled SVG files into sequences of commands including *MoveTo*, *LineTo*, *Bezier* and *ClosePath*. Then, the task is formulated as an NLP-style sequence prediction challenge. However, after all, SVG is intended to present images, or more accurately, visual information, where the *Bezier* curves play a central role. However, the model is never exposed to the knowledge of *Bezier* curves.

The absence of visual information reduces the difficulty of encoding, decoding and differentiating, at the cost of severely constraining the upper limit of the model. In this context, the upper limit refers to the model performance with sufficient data, precise and accurate labels, as well as some currently unknown highly powerful model architecture. For instance, if we want a potent LLM to generate SVG files, we will have to explicitly tell it the correspondence between commands and *Bezier* curves,

define image axis for its canvas, and let it paint step-by-step, drawing the outline of the edges of the target object.

Consequently, our answer to the raised question is: **likely not**. We expect more efforts and expertise spent in encoding visual information, differentiable modelling of *Bezier* curves, and finally assembling all these parts into a robust deep-learning based SVG generator.

6 Conclusion and Future Work

Based on DeepSVG, we have trained a model that generates representative English fonts in SVG format and tried to transplant the model to our own dataset. In the dataset, we collected abundant SVG images of theme clothes, weather and weapons, and then preliminarily modified the formats to make the SVG images correctly rendered and refined the SVG images under regulations. Despite the final results are far from satisfactory, we have reached the following conclusions with brief reasoning.

- (Section. 4.2.1) The complexity of dataset arises from significant intra-class variations, super-hard samples and complicated textures.
- (Section. 4.2.2) The SVG generation fails due to intra-class variations. The solution lies in more refined and precise labelling.
- (Section. 5) The solution to SVG generation is very likely to require visual encoding and knowledge of *Bezier* curves, which *DeepSVG* haven't taken into account.

We have reviewed the whole process of our work in generateing SVG images representing complicated themes and found potential improvement. Throughout our exloration, we have encountered numerous challenges and gained a wealth of knowledge.