Towards Layer-wise Image Vectorization

Xu Ma[1], Yuqian Zhou[2,3*], Xingqian Xu[2*], Bin Sun[1],
Valerii Filev[4], Nikita Orlov[4], Yun Fu[1], Humphrey Shi[2,4]
[1]Northeastern University, [2]UIUC, [3]Adobe Research [4]Picsart AI Research (PAIR)

Figure 1. Examples of the learning course of our Layer-wise Image Vectorization. The proposed method can reconstruct the image in a layer-wise coarse-to-fine manner with only few paths. "N" indicates the path number.

Abstract

Image rasterization is a mature technique in computer graphics, while image vectorization, the reverse path of rasterization, remains a major challenge. Recent advanced deep learning-based models achieve vectorization and semantic interpolation of vector graphs and demonstrate a better topology of generating new figures. However, deep models cannot be easily generalized to out-of-domain testing data. The generated SVGs also contain complex and redundant shapes that are not quite convenient for further editing. Specifically, the crucial layer-wise topology and fundamental semantics in images are still not well understood and thus not fully explored. In this work, we propose Layer-wise Image Vectorization, namely LIVE, to convert raster images to SVGs and simultaneously maintain its image topology. LIVE can generate compact SVG forms with layer-wise structures that are semantically consistent with human perspective. We progressively add new bézier paths and optimize these paths with the layer-wise framework, newly designed loss functions, and component-wise path initialization technique. Our experiments demonstrate that LIVE presents more plausible vectorized forms than prior works and can be generalized to new images. With the help of this newly learned topology, LIVE initiates human editable SVGs for both designers and other downstream applications. Codes are made available at https://github.com/Picsart-AI-Research/LIVE-Layerwise-Image-Vectorization.

1. Introduction

Scalable Vector Graphics (SVGs) [23], which describe images with a collection of parametric shape primitives, have recently attracted increasing attention due to the high practical value in computer graphics. Compared with raster images that present the visual concepts using ordered pixels, vector images enjoy many advantages, like compact file size and resolution-independency. Most importantly, vector

*Yuqian and Xingqian contributed equally.

images provide layer-wise topological information, which is crucial for image understanding and editing.

In the last few years, we have witnessed various achievements in image-to-vector translation [3, 7, 14, 16, 24, 29], mainly due to advances in two technical directions: building powerful generation models, and employing decent differentiable rendering methods. These methods, despite their promising vectorization and generation ability, have always overlooked the topological information hidden behind the raster images. The missing of such information always incurs inadequate learning of vectorization and requires superfluous shape primitives to make up [14, 15, 38]. Some methods attempt to resolve this dilemma by either focusing on particular simple datasets [24, 25] or employing a segmentation pre-processing method [7, 8], but each one has its own drawbacks and subtleties. The first line of work learns to explore the geometric information of fonts or emojis but cannot be generalized to broad domains. The other line that considers segmentation pre-processing method requires heavy pre-processing operations and would segment high-contrast texture into multiple small regions, resulting in redundancy [8]. Hence, a simple yet effective method is desired in the community to capture the layer-wise representation for image-to-vector translation.

In this paper, we introduce a **L**ayer-wise **I**mage **VE**ctorization method, termed as LIVE, to translate a raster image to vector graphics (*i.e.*, SVG) with layer-wise representation. Different from previous works [15, 24], LIVE is model-free and requires no shape primitive labels. This property helps us to escape the regime of particular domains like fonts and emojis, and bypass the difficulty of SVG dataset collection or generalization. Moreover, LIVE enjoys an intuitive and succinct learning course. In each step, we are in pursuit of maximizing the topology exploration rather than only minimizing the pixel-wise difference. The key insight behind this idea is that simply minimizing the vectorization error (*e.g.*, MSE loss between an input raster image and rendered vector graphics) for optimization would lead to a color mean error. We achieve this by a component-wise path initialization method and a novel Unsigned Distance guided Focal loss function (UDF loss). Besides, to mitigate the self-interaction issue, which always occurs in the optimization course [24], we present a novel Self-Crossing loss (Xing loss) by adding constraints to the control points optimization.

We evaluate our proposed method for various tasks, including image-to-vector translation and interpolation across domains (*e.g.*, cliparts, emojis, photos, and natural images) to showcase the effectiveness of LIVE. Our main contributions in this work can be summarized as follows:

- We propose LIVE, a general image vectorization pipeline that hierarchically optimizes the vector graph in a layer-wise manner. Our rendering scheme is fully differentiable and can generate layer-wise SVGs which are largely consistent with human perception.
- Together with LIVE, we also introduce a general initialization method and novel loss functions, including Self-Crossing loss (Xing loss) and Unsigned Distance guided Focal loss (UDF loss). These methods improve the generation of SVGs from raster images, reducing curve intersection and minimizing the shape distortion.
- Comprehensive experiments demonstrate that LIVE can generate precise and compact SVGs in various domains. Our SVG results surpass the results from prior works in terms of simplicity and layer-wise topology.

2. Related Work

In this section, we mainly summarize prior approaches and introduce works that are closely related to our paper.

2.1. Rasterization and Vectorization

Rasterization and vectorization are dual problems in computer graphics. In the past decades, many rasterization works focused on either effective rendering [9, 11, 19, 20] or anti-aliasing [2, 4, 6, 18]. Traditional vectorization methods [5, 13, 28, 32–34] pre-segmented images before vectorization. Among them, [28] and [5] utilized the empirical two-stage algorithm to regress segmented components as polygons and bezigons. Researchers also investigated other approaches that were independent of segmentation, such as diffusion curves [21, 35, 37] and gradient meshes [31]. The rise of deep learning motivated researchers to tackle vectorization via differentiable rendering. Yang *et al*. [36] proposed that bezigons can be directly optimized with self-crafted loss functions by computing gradients using wavelet rasterization [18]. Li *et al*. [14] found shape gradients by differentiating the formula of Reynolds transport theorem [26] with Monta-Carlo edge sampling. Meanwhile, combining differentiable rendering techniques with deep learning models is a trending research direction. New networks that based on recurrent neural network [10], variational autoencoder [10, 17], and transformer [27] are introduced to tackle vectorization and vector graph generation problems. In [10], Ha *et al*. introduce SketchRNN, which is the first RNN-based sketch generation network. In [17], Lopes *et al*. introduce an SVG decoder and combine it with a pixel VAE to generate novel font SVGs in latent space. In [27], Rebeiro *et al*. propose Sketchformer, a transformer-based network that recovers sketches from raster images.

2.2. Image Topology

A human editable SVG should be well-organized in objects and shapes. Prior works have explored such image topology for both raster images and vectorized shapes. A

prototype work was Photo2ClipArt [8], where images were first split into segments which were later vectorized then combined for visual hierarchy. Similar design repeatedly occurs at other research works such as [29, 30]. Nevertheless, these methods were largely relied on the accuracy of the segmentation step and were clumsy to recover implicit shape geometry for complex scenes. Another research branch designed end-to-end frameworks to generate or edit image hierarchy through one forward pass. For example, DeepSVG [3] used VAE as its primary structure where input strokes are first represented via an encoder and later replaced by resampled strokes via a decoder. However, DeepSVG performed SVG to SVG translation, a relatively simple task. Stylized Neural Painting [38] reconstructed images progressively with stylized strokes. Their design principle was to greedy-search for the best stroke to minimize the loss. Yet their main focus was on raster images, not SVG. Lastly, Im2Vec [24] proposed the Encoder-RNN-Rasterizer pipeline to vectorize an image and obtain its topology simultaneously. However, the ordering of the generated shapes was not robust, and the method was domain-specific. Different from aforementioned methods, our LIVE requires no pre-segmentation and no deep models, but exhibits gratifying ability to explore image topologies.

Algorithm 1: Algorithm of LIVE

P = [] ; // list of path control points
C = [] ; // list of path colors
w =1.0 ; // pixel-wise loss weight
$\alpha,\ \beta = 1.0,\ 0.01$; // learning rate
for *n in* $\mathbb{N}$ **do**
  // new points $p \in \mathbb{R}^{n\times 4s\times 2}$
  // new colors $c \in \mathbb{R}^{n\times 4}$
  p, c = init (n, w) ;
  P = concat([P; p]);
  C = concat([C; c]);
  for $j = 1$ *to* t **do**
    $\hat{I} = \text{render}\,(\mathrm{P}, \mathrm{C})$;
    $\mathcal{L} = \mathcal{L}_{\text{UDF}}\left(I - \hat{I}\right) + \lambda \mathcal{L}_{\text{Xing}}\,(P)$;
    $\mathrm{P} = P - \alpha \frac{d\mathcal{L}(\mathrm{P})}{d\mathrm{P}}$; // update points
    $\mathrm{C} = C - \beta \frac{d\mathcal{L}(\mathrm{C})}{d\mathrm{C}}$; // update colors
  end
  $w = \left\| I - \hat{I} \right\|_2$; // update w
end
Output: Scalable Vector Graphic SVG$\{\mathrm{P}, \mathrm{C}\}$.

3. LIVE: Layer-wise Image Vectorization

3.1. Framework

We present a new method to progressively generate an SVG that fits the raster image in a layer-wise fashion. Given an arbitrary input image, LIVE recursively learns the visual concepts by adding new optimizable closed bézier paths and optimizing all these paths. While various shape primitives are available to be appended to an SVG, we consider the parametric closed bézier path as our fundamental shape primitive, like the implementation in [14,24]. There are several reasons behind this setting. First, this strategy would greatly reduce the design space and significantly ease the learning course of LIVE. Also, bézier paths are powerful and easy to approximate diverse shapes, making it unnecessary to introduce various shape primitives. Last, it is convenient for us to control the shape complexity by varying the number of segments s in each path. For complex visual concepts, we can easily increase the segment number to better reconstruct input, and vice versa. Note that the rendering operation is usually non-differentiable, making it difficult to directly optimize the path under the only supervision of the target raster image. To grapple with this dilemma, we take the advantage of a differentiable renderer from [14].

Algorithm. 1 shows the entire pipeline. Briefly, we first introduce a component-wise initialization method that selects the major components as the initialization points. Then we run a recursive pipeline to progressively add n paths according to a path number scheduler sequence $\mathbb{N}$. For each step, we optimize the graph based on some newly proposed objective functions, including an Unsigned Distance guided Focal (UDF) loss and a Self-Crossing (Xing) loss for a better optimization result regarding the reconstruction quality and self-interaction problem. In addition to the layer-wise representation ability, our method is able to reconstruct an image using minimal number of bézier paths, significantly reducing the SVG file size compared to other methods. More details are covered in the following sections.

3.2. Component-wise Path Initialization

We find the initialization of bézier path is crucial in LIVE. A bad initialization will lead to unsuccessful topological extraction and generate redundant shapes. To overcome this defect, we introduce the component-wise path initialization, which greatly helps the optimization course.

The design principle of the component-wise path initialization is to identify the most suitable initial location of the path based on the color and size of each component. One component is one connected area that has a uniform-filled color. As we mentioned earlier, LIVE is a progressive learning pipeline. Given the SVG output from previous stages, we prioritize our next learning target so that the component is both large and missing. We justify such component via the following steps: a) We compute the l_1 pixel-wise color difference between the current rendered SVG and the ground truth image. b) We reject color differences that are smaller than a preset threshold c_α. Empirically, $c_\alpha = 0.1$ in

our paper. Pixel regions with color differences smaller than c_α are considered to be correctly rendered. c) For other regions, we equally quantify all valid color difference values larger than c_α into 200 bins. The quantization is approximately uniformly distributed. d) Finally, we identify the largest connected component based on the quantization, and we then use its center of mass as our next path initial location. If we want to add K more paths, then we choose the top-K components for next-stage initialization. Note that for each path, we consider the circle initialization method that all control points are initialized uniformly on a circle [24]. Empirically, this simple strategy helps to ease the optimization course and is proved to be helpful.

The merit of our component-wise path initialization is that it maintains a good balance between the color and size of the missing region. Unlike DiffVG [14] and Neural Painting [38], in which the former randomly initializes paths and the later initializes strokes based on MSE, our approach focuses on semantic-influencing components that are independent from its RGB value. While adding new paths to the existing figures, our initialization methods can always identify the largest missing components with similar color, and fill in the major regions.

3.3. Loss Function

3.3.1 UDF Loss for Reconstruction

In previous work [14, 24, 25], a commonly used loss function to minimize the error between target image $I \in \mathbb{R}^{w\times h\times 3}$ and rendered output $\hat{I} \in \mathbb{R}^{w\times h\times 3}$ is the mean square error (MSE) $\left\|I-\hat{I}\right\|_2^2$, where 3 represents RGB and $w \times h$ represents image size. MSE loss is simple yet efficient for image comparison, but it will bias towards the mean color of the entire target image, as shown in Figure 2. This phenomenon is because MSE are computed using all available pixels, while not all pixels are related to the optimizing path. Hence, we are encouraged to only focus on valid pixels and ignore the unrelated ones.

To resolve this problem, we introduce the Unsigned Distance guided Focal (UDF) loss, which treats each pixel differently based on the distance to the shape contour. Intuitively speaking, UDF loss emphasizes the differences close to the contour and suppresses differences in other locations. By doing so, LIVE protects itself from MSE's mean color issue and therefore maintains accuracy color reconstruction.

Without losing generality, we formulate our UDF loss assuming the case with a single path. We render the path and compute each pixel's signed distance to the path represented by $d_i, i \in \{1, ..., h \times w\}$. We then threshold, flip, and normalize the unsigned distance $|d_i|$ by:

$$d_i' = \frac{\text{ReLU}(\tau - |d_i|))}{\sum_{j=1}^{w\times h} \text{ReLU}(\tau - |d_j|))}, \tag{1}$$

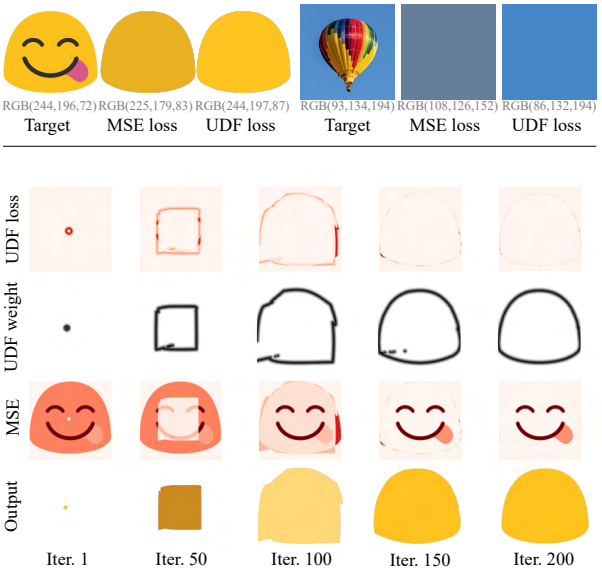

Figure 2. **Top line** shows the differences between UDF loss and MSE loss when learning the first path. MSE loss is biased towards the mean color of the target image, while our UDF loss preserves the color of the target shape. Best viewed in color. **Bottom block** presents an example of UDF loss optimization course for the first path. We normalize all values to the range of $[0, 1]$ for better visualization. Darker color (gray or red) means a higher value.

where both i and j are indices of pixels and τ is a distance threshold. We set τ equals to 10 by default. Next, we formulate our Unsigned Distance guided Focal loss as

$$\mathcal{L}_{\text{UDF}} = \frac{1}{3}\sum_{i=1}^{w\times h} d_i' \sum_{c=1}^{3}\left(I_{i,c} - \hat{I}_{i,c}\right)^2, \tag{2}$$

where i indexes the pixel in I and c indexes the RGB channel. With the help of UDF loss, we are able to pay close attention to the path contour and to avoid the effect from inner or distant regions. Figure 2 shows the learning course of Unsigned Distance guided Focal loss. To support multiple paths in our LIVE framework, we can easily extend Equation 2 by averaging d_i' over all paths.

3.3.2 Xing Loss for Self-Interaction Problem

We notice that it is possible for some bézier paths to become self-interacted during the course of optimization, leading to detrimental artifacts and improper topology [24, 36]. While it might be expected that additional paths can cover the artifacts, we emphasize this would complicate the generated SVG and cannot effectively explore the underlying topological information. To this end, we introduce the self-interaction (Xing) loss to mitigate this problem.

Assuming all the bézier curves in our paper are third-order, by analyzing a number of optimized shapes, we found

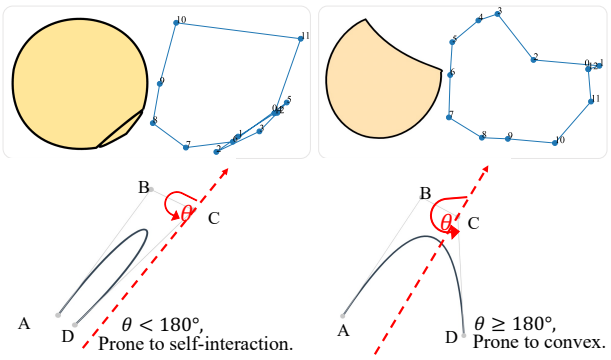

Figure 3. Illustration of self-interaction problem. **Top left pair** shows a circle with self-interactions and the lines between its adjacent control points. **Top right pair** shows a shape without self-interaction. **Bottom line** illustrates our Xing loss. In a cubic bézier path, we encourage the angle (θ) between the first ($\vec{AB}$) and the last ($\vec{CD}$) control points connections greater than $180°$.

that a self-interacted path always intersects the lines of its control points, and vice versa. Figure 3 shows the examples. This suggests that instead of optimizing the bézier path, one potential solution would be adding a constraint on the control points. Assume the control points of a cubic bézier path are A, B, C, and D in sequence, we add a constraint that the angle between $\overrightarrow{AB}$ and $\overrightarrow{CD}$ (θ in the figure) should be greater than $180°$. We first determine the characteristic (acute angle or obtuse angle) of $\angle ABC$ as D_1 and the value of $\sin(\theta)$ as D_2 by

$$D_1 = \mathbb{I}\left(\vec{AB} \times \vec{BC}\right), \quad D_2 = \frac{\vec{AB} \times \vec{CD}}{\left\|\vec{AB}\right\| \left\|\vec{CD}\right\|}, \tag{3}$$

where $\mathbb{I}(\cdot)$ is a sign function that returns 1 (if $D_1 > 0$) or 0 (if $D_1 \leq 0$), $\times$ is vector production that returns a real value. We then formulate our Xing loss as

$$\mathcal{L}_{\text{Xing}} = D_1 \left(\text{ReLU}\left(-D_2\right)\right) + \left(1 - D_1\right) \left(\text{ReLU}\left(D_2\right)\right). \tag{4}$$

The basic idea of Equation 4 is that we only optimize the case when $\theta < 180°$ (achieved by $\text{ReLU}(\pm D_2)$). The first term is designed for case $D_1 = 1$ and the second term is designed for case $D_1 = 0$. Combining both UDF loss and Xing loss, our final loss function $\mathcal{L}$ is given by

$$\mathcal{L} = \mathcal{L}_{\text{UDF}} + \lambda \mathcal{L}_{\text{Xing}}, \tag{5}$$

where λ is set to 0.01 empirically to balance the two losses.

3.4. Datasets

Existing vector graphics datasets [3, 16] mainly focus on the generation of either fonts or icons, but a broader domain of images is not explored. Also, there is no testing set that can serve as a benchmark for evaluation. In this paper, we test our model on two datasets, an **Emoji** dataset that mainly collects a subset of emojis from [1], and a **Pics** dataset that collects images from different domains. Figure 4 showcases some examples from Emoji and Pics datasets.

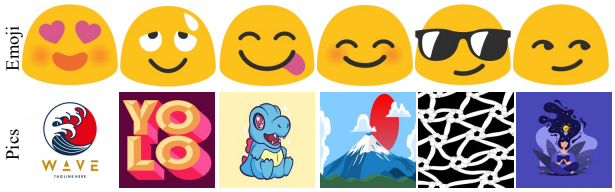

Figure 4. Exemplars from Emoji dataset and Pics dataset.

Emoji Dataset. We collect 134 emojis with various shapes, colors, and combinations from the NotoEmoji project [1]. While various fonts and icons are given in this project, we mainly collect the smiling face images, and resize all collected images to a resolution of 240×240. Compared with the emojis used in [24], our Emoji dataset includes more images and presents more diversity. Since images in [1] are relatively simple and present clear topological information, we mainly use this dataset to evaluate the exploration of layer-wise representation.

Pics Dataset Besides the Emoji dataset, we also introduce Pics dataset, which contains 153 images, including fonts, icons, and complex clipart images. Compared with the Emoji dataset, the Pics dataset is more complex and challenging for image vectorization. Moreover, some images in the Pics dataset are with various backgrounds, further increasing the vectorization difficulty. We mainly use this dataset to examine the layer-wise modeling and compact SVG with fewer paths.

Note that our LIVE is a model-free method, both datasets are only used to evaluation. Besides the two datasets, we also evaluate LIVE on some realistic photos.

3.5. Implementation Details

We implement LIVE in PyTorch [22] and optimize it using Adam optimizer [12], with a learning rate of 1 and 0.01 for points and colors optimization, respectively. By default, we use four segments for each path in our experiments. The circle radius is set to 5 pixels for circle initialization. For each optimization step, all the parameters are trained for 500 iterations. Since our method progressively adds new paths to the canvas, the number of new paths in each step is flexible. Considering both the efficiency and vectorization quality, we set the path number in ith optimization step to be $\min\left(2^{i-1}, 32\right)$. Other number setting strategies also work, like adding one path each time or a customized setting.

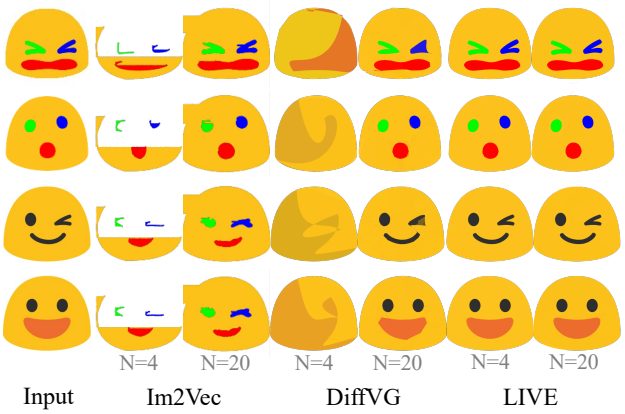

Figure 5. Qualitative reconstruction comparison. We compare LIVE with Im2Vec and DiffVG using a different number of paths. We select four paths (the number of components in each image) and 20 paths (default value in Im2Vec) for comparison. Intuitively, LIVE can achieve perfect results using only four paths, and more paths would not degrade the performance.

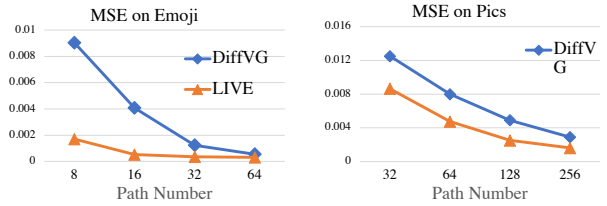

Figure 6. MSE *vs.* path number on Emoji dataset and Pics dataset. Our LIVE achieves much better reconstruction results than DiffVG, especially when the path number is smaller.

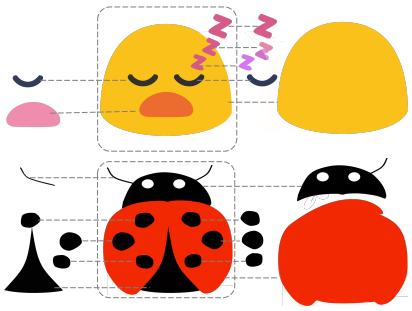

Figure 7. Illustration of the layer-wise representation on emoji and clipart images. When the visual clues are easy to model, LIVE can directly model each individual component, presenting a reasonable and clear layer-wise representation.

4. Experiments

4.1. Vectorization Quality

We first evaluate LIVE's vectorization quality with both quantitative and qualitative analysis, measuring the differences between input targets and SVG rendered images.

Qualitative Comparisons. Figure 5 shows the visual comparisons with previous state-of-the-art methods including DiffVG [14] and Im2Vec [24]. For fairness, we set the number of path to 4 (the number of components in these emojis) and 20 (the default setting in Im2Vec) for evaluation. Clearly, our LIVE achieve a more faithful reconstruction with better component shapes and colors, while others may still have other artifacts. Therefore, the proposed LIVE better decouples the geometry of different components. More results are in the supplementary materials.

Quantitative Results. Next, we quantize the vectorization results on the Emoji and Pics datasets. For a fair comparison, the number of segments is set to 4 as the default setting in DiffVG. To showcase that LIVE can reconstruct one image with a minimal number of paths, we vary the path number from 8 to 64 for the simple Emoji dataset and from 32 to 256 for the complex Pics dataset. For comparison, we calculate the MSE of each image for the entire dataset.

The results on Emoji and Pics benchmarks are reported in Figure 6. Clearly, LIVE shows a much lower MSE than DiffVG, especially when the path number is small. When only a few paths are employed, LIVE is able to fit the desired shapes, leading to a better result. Increasing excessive paths would saturate the vectorization performance.

4.2. Layer-wise Representation

Besides the vectorization quality and efficiency, the main objective of LIVE is to build a Layer-wise representation. Empirically, LIVE is able to explicitly vectorize each individual visual concept and explore the layer-wise representation for simple images like emojis and simple cliparts. We demonstrate the layer-wise representation ability of LIVE in Figure 7. As shown in the figure, each component is clearly learned as a single bézier path. Different from the vectorization methods that leverage segmentation pre-processing or use abundant paths, we can learn each component as an unbroken shape. In Figure 9, we compare the vectorization results of LIVE and DiffVG on the Emoji benchmark.

For complex images like photos and natural images, the topological clues are relatively hard to model. However, LIVE still exhibits a gratifying ability of the "coarse-to-fine" learning style, as shown in Figure 8. LIVE is more likely to achieve better reconstruction performance under the same number of paths. Moreover, we notice that LIVE models the local information much better than others, as shown in the red boxes. This may be explained by our progressive learning and the initialization method. In each step, LIVE encourages the new paths to fit the local details. While previous paths have successfully reconstructed the main context, the newly added path will only focus on ini-

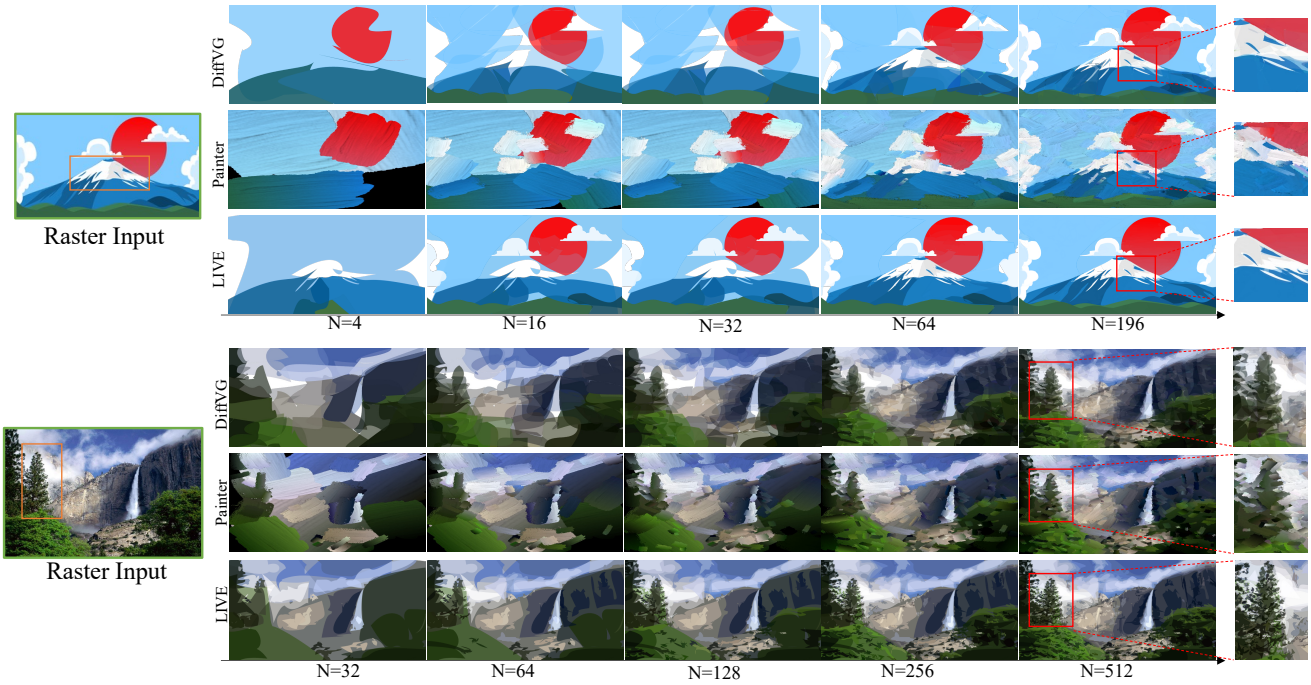

Figure 8. We showcase the results of DiffVG [14], Neural Painting [38] and our LIVE under different number of paths/strokes. The two images are taken from the Pics dataset and testing images from [38], respectively. *Note that Neural Painting is not only designed for reconstruction. We still visually compare with it because of its progressive learning fashion, which is similar to our LIVE. We use red boxes to emphasize the differences.* Please zoom in to see the details. More results will be presented in the supplementary materials.

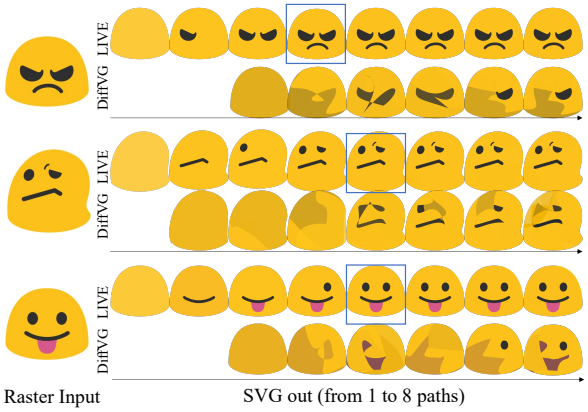

Figure 9. Vectorization results of DiffVG and LIVE. LIVE explicitly vectorized each visual concept, without any redundancy and artifacts. Blue boxes indicate when LIVE vectorized all concepts, and adding more paths will not damage the results.

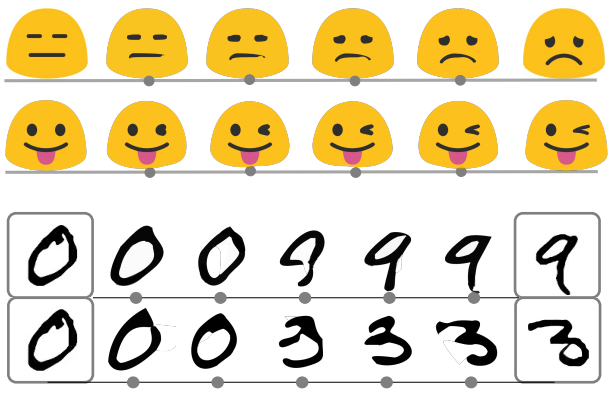

Figure 10. Two examples of interpolation. **Top tow rows** showcase the results of linear interpolating the bézier control points between two generated SVGs. **Bottom two rows** show the results of combining LIVE and a simple VAE. Gray boxes mark the input raster images. Intermediate images indicate the interpolations.

tialized local regions with the enforcement of UDF loss. A comprehensive user study also demonstrated the superiority of LIVE (please refer to the supplementary).

4.3. Interpolation

Among existing vectorization methods, some VAE-based methods explored the application of interpolation [3, 16, 24]. Even our LIVE is not based on the VAE model, we showcase that it is easy to achieve interpolation by integrating with vanilla raster image-based VAE model.

Before implementing the VAE interpolation, we first conduct an interesting interpolation experiment: given two semantically similar SVGs generated by LIVE, we directly interpolate the control points of each ordered path linearly. Normally, two SVGs are hard to be interpolated due to the

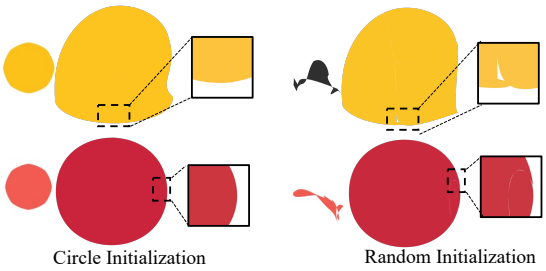

Figure 11. Examples of different initialization methods. For each triplet, we show the initialization (first column), output(second column), and the detail (third column). Zoom in to see better.

disorder of both shapes and control points. In contrast, our LIVE will not suffer from this issue because of the ordered topology structures after optimization. Empirically, even with the simple interpolation of linear control points, LIVE still presents a reasonable result as shown in Figure 10.

Next, we integrate our method with the VAE model. We train a simple VAE model on the MNIST dataset. Next, two random images are selected, we linearly interpolate the two latent vectors to obtain the interpolated images and use our LIVE to vectorize the resulted images sequence. To form a continuous sequence, we treat the previous result as the initialization of the next sample. Results in Figure 10 demonstrate that combing with a vanilla VAE model, our method works for interpolation as well. Given the efficient optimization method and great generalization ability, LIVE can be more practical to achieve the interpolation goal when combined with a powerful image generation model.

4.4. Ablation study

Circle Initialization. We first investigate the effectiveness of control point initialization. Figure 11 compares the circle initialization and random initialization. Clearly, the circle initialization significantly reduces the artifacts compared with random initialization. Moreover, we notice that circle initialization is more likely to achieve better vectorization results, as shown in the first row. The reason is that with a circle initialization of control points, the close path is enforced to be convex, and gets a finer optimization result.

Xing Loss. To understand the effectiveness of the proposed Xing loss, we conduct an ablation study to investigate the impact of Xing loss through visualization in Figure 12. With the help of Xing loss, we clearly mitigate the problem of self-interaction under the same optimization conditions. The circle shape tends to not intersect, given the constraints on the control points. It shows the proposed Xing loss is an intuitive, simple but effective objective function for mitigating the self-intersection issues. More results will be presented in the supplementary materials.

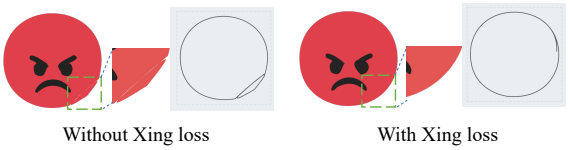

Figure 12. Illustration of the effectiveness of Xing loss. Each triplet shows the generated SVG, details, and the stroke of the face. By adding Xing loss, we greatly mitigate the self-interaction problem. Please zoom in to see the details.

4.5. Discussion

Limitations and Future Works. LIVE presents a layer-wise vectorization result, which can be used for further clipart creation or other applications. However, there are still some issues that we can discuss. First, the layer-wise operation is not efficient as the single-pass optimization. Some other methods also suffer from this issue [38]. An interesting research direction would be how we can combine the highly-efficient inference of deep models with the generalization ability of the optimization-based methods. Second, introducing gradient color and adaptively choosing the segment numbers and color type for each segment will be worth exploring. Third, for more complicated images like landscape or human photos, combining layer-wise vectorization with deep amodal segmentation in pixel space will be an interesting topic. We leave those for future works.

Potential Negative Impact. Image to vector technologies can be misused by illegally converting and copying vector graph resources online, especially for easily reused and modified font or other images. To mitigate these issues, one can protect the copyright of the graph by using watermarks on raster images. Besides, though our paper achieves reasonable layer-wise modeling of the images, results converted from raster images can still be differentiated by checking whether each component is intact enough. Those actions will avoid the abuse of similar algorithms.

5. Conclusion

In this work, we present Layer-wise Image VEctorization (LIVE), a framework to equip image vectorization with layer-wise representation. LIVE progressively infers the input raster image with the help of component-wise path initialization and new loss functions: an UDF loss for vectorization and a Xing loss to mitigate the self-interaction problem. With LIVE, we can explicitly vectorize individual components for a simple emoji or clipart, and investigate the "coarse-to-fine" representation for complex natural images. To ease the evaluation of image vectorization, we also present two datasets, Emoji, and Pics. Besides image vectorization, LIVE can also be integrated with other methods to explore other applications like interpolation.

References

[1] Note emoji. https://github.com/googlefonts/noto-emoji. Accessed: 2021-09-30. 5

[2] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 251–256, 1991. 2

[3] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. Deepsvg: A hierarchical generative network for vector graphics animation. *arXiv preprint arXiv:2007.11301*, 2020. 2, 3, 5, 7

[4] Robert L Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG)*, 5(1):51–72, 1986. 2

[5] James Richard Diebel. *Bayesian Image Vectorization: the probabilistic inversion of vector image rasterization*. Stanford University, 2008. 2

[6] Mark AZ Dippé and Erling Henry Wold. Antialiasing through stochastic sampling. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 69–78, 1985. 2

[7] Vage Egiazarian, Oleg Voynov, Alexey Artemov, Denis Volkhonskiy, Aleksandr Safin, Maria Taktasheva, Denis Zorin, and Evgeny Burnaev. Deep vectorization of technical drawings. In *European Conference on Computer Vision*, pages 582–598. Springer, 2020. 2

[8] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. Photo2clipart: Image abstraction and vectorization using layered linear gradients. *ACM Transactions on Graphics (TOG)*, 36(6):1–11, 2017. 2, 3

[9] Francisco Ganacim, Rodolfo S Lima, Luiz Henrique de Figueiredo, and Diego Nehab. Massively-parallel vector graphics. *ACM Transactions on Graphics (TOG)*, 33(6):1–14, 2014. 2

[10] David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017. 2

[11] Mark J Kilgard and Jeff Bolz. Gpu-accelerated path rendering. *ACM Transactions on Graphics (TOG)*, 31(6):1–10, 2012. 2

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 5

[13] Gregory Lecot and Bruno Levy. Ardeco: Automatic region detection and conversion. In *17th Eurographics Symposium on Rendering-EGSR'06*, pages 349–360, 2006. 2

[14] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020. 2, 3, 4, 6, 7

[15] Songhua Liu, Tianwei Lin, Dongliang He, Fu Li, Ruifeng Deng, Xin Li, Errui Ding, and Hao Wang. Paint transformer: Feed forward neural painting with stroke prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6598–6607, 2021. 2

[16] Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. A learned representation for scalable vector graphics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7930–7939, 2019. 2, 5, 7

[17] Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. A learned representation for scalable vector graphics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7930–7939, 2019. 2

[18] Josiah Manson and Scott Schaefer. Wavelet rasterization. In *Computer Graphics Forum*, volume 30, pages 395–404. Wiley Online Library, 2011. 2

[19] Diego Nehab and Hugues Hoppe. Random-access rendering of general vector graphics. *ACM Transactions on Graphics (TOG)*, 27(5):1–10, 2008. 2

[20] Peter Nilsson and David Reveman. Hardware accelerated image compositing using opengl. In *Proc of Usenix*, volume 4. 2

[21] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: a vector representation for smooth-shaded images. *ACM Transactions on Graphics (TOG)*, 27(3):1–8, 2008. 2

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019. 5

[23] Antoine Quint. Scalable vector graphics. *IEEE MultiMedia*, 10(3):99–102, 2003. 1

[24] Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. Im2vec: Synthesizing vector graphics without vector supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7342–7351, 2021. 2, 3, 4, 5, 6, 7

[25] Pradyumna Reddy, Zhifei Zhang, Matthew Fisher, Hailin Jin, Zhaowen Wang, and Niloy J Mitra. A multi-implicit neural representation for fonts. *arXiv preprint arXiv:2106.06866*, 2021. 2, 4

[26] Osborne Reynolds. *The sub-mechanics of the universe*, volume 3. University Press, 1903. 2

[27] Leo Sampaio Ferraz Ribeiro, Tu Bui, John Collomosse, and Moacir Ponti. Sketchformer: Transformer-based representation for sketched structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14153–14162, 2020. 2

[28] Peter Selinger. Potrace: a polygon-based tracing algorithm. *Potrace (online), http://potrace. sourceforge. net/potrace. pdf (2009-07-01)*, 2, 2003. 2

[29] I-Chao Shen and Bing-Yu Chen. Clipgen: A deep generative model for clipart vectorization and synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 2021. 2, 3

[30] Wataru Shimoda, Daichi Haraguchi, Seiichi Uchida, and Kota Yamaguchi. De-rendering stylized texts. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1076–1085, 2021. 3

[31] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics (TOG)*, 26(3):11–es, 2007. 2

[32] Daniel Sỳkora, Jan Buriánek, and Jirí Zára. Sketching cartoons by example. In *SBM*, pages 27–33, 2005. 2

[33] Daniel Sỳkora, Jan Buriánek, and Jiří Žára. Video codec for classical cartoon animations with hardware accelerated playback. In *International Symposium on Visual Computing*, pages 43–50. Springer, 2005. 2

[34] Tian Xia, Binbin Liao, and Yizhou Yu. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Transactions on Graphics (TOG)*, 28(5):1–10, 2009. 2

[35] Guofu Xie, Xin Sun, Xin Tong, and Derek Nowrouzezahrai. Hierarchical diffusion curves for accurate automatic image vectorization. *ACM Transactions on Graphics (TOG)*, 33(6):1–11, 2014. 2

[36] Ming Yang, Hongyang Chao, Chi Zhang, Jun Guo, Lu Yuan, and Jian Sun. Effective clipart image vectorization through direct optimization of bezigons. *IEEE transactions on visualization and computer graphics*, 22(2):1063–1075, 2015. 2, 4

[37] Shuang Zhao, Frédo Durand, and Changxi Zheng. Inverse diffusion curves using shape optimization. *IEEE transactions on visualization and computer graphics*, 24(7):2153–2166, 2017. 2

[38] Zhengxia Zou, Tianyang Shi, Shuang Qiu, Yi Yuan, and Zhenwei Shi. Stylized neural painting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15689–15698, 2021. 2, 3, 4, 7, 8