

M. Dziuba, I. Jarsky, V. Efimova, A. Filchenkov

IMAGE VECTORIZATION: A REVIEW

ABSTRACT. Nowadays, there exist many diffusion and autoregressive models that show impressive results for generating images from text and other input domains. However, these methods are not intended for ultra-high-resolution image synthesis. Vector graphics are devoid of this disadvantage, so the generation of images in this format appears to be a very promising direction. Instead of generating vector images directly, one can first synthesize a raster image and then apply vectorization. Vectorization is the process of converting a raster image into a similar vector image using primitive shapes. Besides being similar, the generated vector image is also required to contain a minimal number of shapes for rendering. In this work, we focus specifically on machine learning-compatible vectorization methods. We consider Mang2Vec, Deep Vectorization of Technical Drawings, DiffVG, and LIVE models. We also provide a brief overview of existing online methods. We also recall other algorithmic methods, Im2Vec and ClipGEN models, but they do not participate in the comparison, since there is no open implementation of these methods or their official implementations do not work correctly. Our research shows that despite the ability to directly specify the number and type of shapes, existing machine learning methods take a very long time and do not accurately recreate the original image. We believe that there is no fast universal automatic approach and human control is required for every method.

§1. INTRODUCTION

In computer graphics, two main approaches for image representation coexist. While a bitmap image is a matrix of pixels, a vector image is a sequence of shapes drawn on a canvas. Raster graphics are commonly used for complex images containing a large number of visual details and complex color transitions. Most often these are photos and photorealistic drawings. At the same time, vector images consist of shapes that typically have a constant one-color fill. This results in the simplicity and abstractness of the resulting image. Therefore, the primary domains of vector graphics are icons, logos, simple illustrations, and fonts. Vector images are easily

Key words and phrases: vector graphics, image vectorization, computer vision.

embedded in HTML markup, and a crucial requirement is the small size of the code describing them for fast data transfer to the user and subsequent rapid rendering. The most popular vector format is SVG, which defines a vector image as a tag sequence using XML markup. Each tag can use XML attributes to specify the shape color characteristics and transformations. Using this markup, a renderer program draws an image consisting of the specified figures. Thus, the task of vectorizing a bitmap image is similar to the task of obtaining a sequence of shapes and their parameters that together form the original raster image.

In 2014, generative adversarial models [9] became the best machine learning algorithms for image synthesis. Since then, image synthesis has become an important part of digital art, data augmentation techniques, design, fashion, and several other domains. Currently, the image generation task is solved with deep generative models based on diffusion models [24,25] and autoregressive models [7,35].

Modern generative methods have recently achieved significant success in the raster domain. However, despite these approaches being designed to generate highly realistic images in different styles from input text, the resulting images do not have a high resolution; usually, it is less than 2048x2048 pixels. This, however, is not enough for logos, covers, and for printing. In these domains, vector graphics are standard.

One of the ways to obtain such images is to use vector graphics instead of raster graphics. Notably, little research is being done in vector image generation [1, 4, 5, 8, 11, 19, 26, 32]. The creation of vector images is still done by humans, but working with vector image code is not an easy task. Therefore, the ability to generate such images automatically with a minimal number of post-processing steps is a highly relevant task.

A possible solution for obtaining a vector image is the vectorization of raster images. It may also allow one to enrich vector datasets that are required for training vector image generation algorithms, but the size of which is still insufficient in comparison with bitmap image datasets.

Existing image vectorization methods can be divided into two categories: algorithmic and machine learning-based methods. Algorithmic approaches have recently been reviewed in [31]. The authors classify vectorization methods into mesh-based and curve-based. Mesh-based methods split the image into non-overlapping 2D patches and interpolate colors across them. The patch shape can be triangular [10, 18, 38], rectangular [15, 21, 29], or even irregular, for example, in the form of Bézigons,

closed regions bounded by Bézier curves [16, 30, 34], and the patch vertices or interior can store color and other attributes. Curve-based methods are based on diffusion curves, which are Bézier curves with colors defined on their left and right sides. Color discontinuities can be modeled by diffusing the colors from both sides of the curves to create the resulting image. For smooth edges, the diffusion process might be followed by a blurring phase. There are different formulations of diffusion curves to work with: basic and harmonic [2, 12, 37], and biharmonic [33] as well.

Although several vectorization methods exist, no decent comparison has been made to the best of our knowledge. In this work, we focus on machine learning-compatible image vectorization methods. We aim to classify and compare them using different evaluation criteria. The main comparison criteria of vectorization methods are: (1) similarity to the original bitmap; (2) simplicity or complexity of the resulting image including the number of shapes and their parameters; (3) generation speed; (4) versatility, i.e., the ability to generate a fairly accurate copy of the input image without prior model training; (5) human control to adjust hyperparameters.

The contributions of this paper include an overview of machine learning-compatible vectorization methods and a comparison of their performance.

§2. MACHINE LEARNING-COMPATIBLE METHODS

2.1. DiffVG. The work presenting the DiffVG method [17] provides grounding for machine learning-based vector image generation methods since it implements a differentiable vector image rasterization function linking the vector and raster domains. A raster image can be vectorized with DiffVG by fitting a predefined number of randomly initialized Bézier curves to the target image, see Fig. 1. Optimization can be performed by minimizing the L_2 loss between a raster and rasterized vector images or a deep-learning-based perceptual loss [36].

The authors also propose a simple variational autoencoder [13] for vectorizing MNIST digits images [3]. The encoder convolves the input raster image into a latent space, a vector image is synthesized from the embedding, then the input and synthesized images are compared using the proposed rasterization function. The resulting images are inaccurate because each character is represented by several curves that look like an artist's strokes. The results do not exactly match the original images even for such a simple dataset as MNIST.

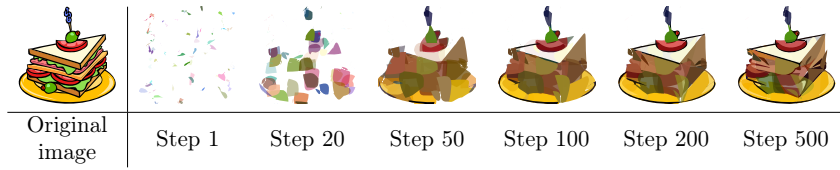


Figure 1. Iterative vectorization using the DiffVG method. Each generated image has 62 paths, and the same number of paths is used in the original vector image.

Another feature of this method is also worth mentioning. The running time of the rasterization algorithm significantly depends on the size of the output image. However, output size reduction results in the loss of important image details. This problem arises due to the nature of vector images. By analogy with decreasing the size of raster images, this leads to the loss of image details, and the same effect occurs when the number of paths is reduced in vector images.

2.2. Im2Vec. The Im2Vec work [23] offers a model for image vectorization and interpolation. Its architecture is based on the variational autoencoder (VAE) [14] proposed in DiffVG. The model maps an input raster image into a latent space and then generates a similar vector image. While training, to compare the generated vector image with the input raster image the vector image is rasterized using a differentiable rasterizer.

In the paper, the authors propose a new way of generating closed shapes. Initially, a circle is sampled for every shape, and then, based on the latent vectors of the shapes generated with LSTM, the circle is deformed.

Since the difference between the target and output images is significant at the beginning of training, the authors suggest using a multi-resolution loss. They propose to rasterize images in different resolutions, thus building a pyramid of images, with the loss function being calculated for every layer. The total multi-resolution loss to be optimized is

$$\mathbb{E}_{I \sim D} \sum_{l=1}^L \|pyr_l(I) - O_l\|^2,$$

where L is the number of pyramid levels, $pyr_l(I)$ is the l -th pyramid level, O_l is the output rasterized at the corresponding spatial resolution, and D is the training dataset.

Unfortunately, the results can hardly be reproduced using the official implementation. During the first 700 epochs of training on the emoji dataset, which was collected and published by the authors, the generation result had no changes and consisted of a single point in the center of the image. We have also noticed that the shapes' colors are fixed in the implementation, i.e., no separate color generation can be performed, so we have added a separate LSTM model for predicting the colors of shapes. However, even after resolving issues to their implementation, the authors themselves point out that color prediction causes instability during generation.

Thus, this work cannot be considered as a universal model for vectorizing any image, because: (1) it must be pretrained on a large number of vector images, which are hard to obtain; (2) the shortcomings we have discussed above are likely to lead to learning instability providing resulting images of poor quality.

2.3. LIVE. The work [20] introduced the LIVE vectorizer. LIVE is a logical continuation of the iterative method proposed in DiffVG. However, LIVE does not operate with all shapes simultaneously from the first iteration. Instead, it gradually adds one or more shapes to the canvas layer by layer and then performs an optimization step.

Unlike DiffVG, LIVE operates only with closed shapes consisting of cubic Bézier curves. There is an issue that some of them may become self-intersecting during optimization, which results in undesirable artifacts and incorrect topology. Although additional paths could cover artifacts, it would complicate the resulting SVG making it impossible to effectively investigate the underlying topological data. The authors discovered that a self-intersecting path always intersects the lines of its control points, and vice versa, assuming that all of the Bézier curves are of the third order. Therefore, the authors introduce a new loss function (Xing-loss) designed to solve this self-intersection problem. The fundamental idea is to only optimize the case when the angle of the curve is 180° degrees. In other words, the authors urge the angle between the first and last control points connections to be greater than 180° in a cubic Bézier path. This loss acts as a regularizer on self-intersection, and its formal definition is

$$\mathcal{L}_{Xing} = D_1(\text{ReLU}(-D_2))(1 - D_1)(\text{ReLU}(D_2)),$$

where D_1 is a characteristic of the angle between two segments of a cubic Bézier path, and $D_2 = \sin \alpha$ is the value of that angle.

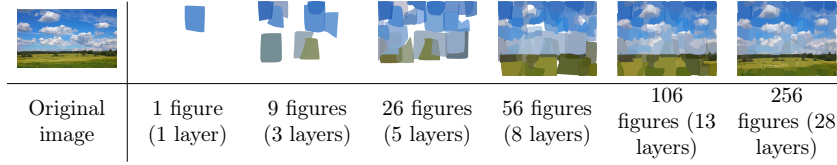


Figure 2. Iterative vectorization using the LIVE method.

To make each path responsible only for a single feature of the image, the authors introduce the Unsigned Distance guided Focal loss (UDF loss) as well; it treats every pixel differently depending on how close it is to the shape contour. According to intuition, the UDF loss amplifies differences near the contour and suppresses differences in other areas: LIVE weighs an L_2 reconstruction loss by the distance to the nearest path. By doing this, LIVE defends against the mean color problem caused by MSE and preserves accurate color reconstruction:

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

where I is the target image, \hat{I} is the rendering, c indexes RGB channels in I , d'_i is the unsigned distance between pixel i and the nearest path boundary, and w, h are the width and height of the image respectively.

LIVE produces relatively clean SVGs by initializing paths in stages, localized to poorly reconstructed, high-loss regions. LIVE’s main advantage is its ability to reconstruct an image with a user-defined amount of paths, significantly reducing the SVG file size compared to other methods. However, it takes a lot of time to vectorize an image even on a GPU, thus this method is hardly applicable in practice for complex images with a large optimal number of paths. Fig. 2 illustrates the iterative process.

2.4. ClipGen. The ClipGen paper [27] proposes a method based on deep learning for automatically vectorizing the clipart of man-made objects. The suggested approach needs a raster clipart image and relevant object category (for instance, airplanes). It sequentially creates new layers, each formed by a new closed path that is filled with a single color. All layers are combined to create a vector clipart that fits the desired category to produce the resulting image. The suggested method is built on an iterative generative model that chooses whether to keep synthesizing new

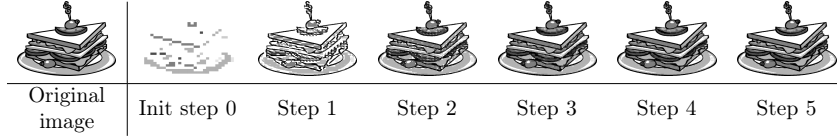


Figure 3. Vectorization using the Mang2Vec method.

layers and defines their geometry and appearance. For training their generative model, the authors develop a joint loss function that includes shape similarity, symmetry, and local curve smoothness losses, as well as vector graphics rendering accuracy loss for synthesizing a human-recognizable clip-art. However, ClipGen only works with a predefined number of categories, therefore it cannot process arbitrary images.

2.5. Mang2Vec. The authors of the Mang2Vec work [28] suggest the first method for vectorizing raster mangas by using deep reinforcement learning. They develop an agent that is trained to generate the best possible sequence of stroke lines while being constrained to match the target manga visual features. The control parameters for the strokes are then collected and converted to vector format. They also propose a reward to produce accurate strokes and a pruning method to avoid errors and redundant strokes. Mang2Vec works only with black and white manga and cannot be used with colored images.

2.6. Deep Vectorization of Technical Drawings. The paper [6] proposes a technical line drawings vectorization method (DVoTD), for example, for drawings of floor plans. The authors convert a technical raster drawing, which is cleared of text, into a set of line segments and quadratic Bézier curves that are specified by control points and width. They preprocess the input image by eliminating noise, modifying contrast, and adding missing pixels. Then, they divide the image into patches and calculate the starting primitive parameters for each patch. To do this, each patch is encoded with a ResNet-based feature extractor and decoded as feature embeddings of the primitives using a sequence of Transformer blocks. To train the network for primitive extraction, the following loss function is proposed:

$$L(p, \hat{p}, \theta, \hat{\theta}) = \frac{1}{n_{prim}} \sum_{k=1}^{n_{prim}} (L_{cls}(p_k, \hat{p}_k) + L_{loc}(\theta_k, \hat{\theta}_k)),$$

where

$$L_{cls}(p_k, \hat{p}_k) = -\hat{p}_k \log p_k - (1 - \hat{p}_k) \log(1 - p_k),$$

$$L_{loc}(\theta_k, \hat{\theta}_k) = (1 - \lambda) \|\theta_k - \hat{\theta}_k\|_1 + \lambda \|\theta_k \hat{\theta}_k\|_2^2,$$

\hat{p} is the target confidence vector (all ones with zeros at the end that indicate placeholder primitives all target parameters $\hat{\theta}_k$ of which are set to zero).

The approximated primitives improve by aligning to the cleaned raster. Improved predictions from all patches are combined.

§3. ONLINE VECTORIZATION METHODS

There are plenty of websites that can vectorize any raster image. Existing online methods can be free to use (svgstorm.com, www.visioncortex.org/vtracer, vectorization.eu) or proprietary (vectorizer.io).

Common options provided by these methods is the selection of vector graphics output file format (SVG, EPS, PDF), color palette, and the number of colors used. Some services allow to choose the quality of image detail (Low, Medium, High), the type of shapes used (Curve Fitting - pixel/polygon/curve), background removal, and many other actions and parameters. These options affect the processing speed, the visual result, and the number of shapes. The generation performance highly depends on the resolution of the input raster image and its details complexity. On average, the processing time of a single image is 10 seconds.

Even though they are easy to use, a lot of parameters should be fixed by a user. It should be mentioned that the resulting image quality and its size significantly depend on the input image quality and resolution. Low quality results in producing images with a lot more paths that are actually needed leading to noticeable artifacts.

One of the popular online services for vectorization is VTracer [22], which provides many options for a user. According to its documentation, the method first clusters the input image by hierarchical clustering and each of the output clusters is traced into vector format. After converting pixels into staircase-like paths, the method then simplifies the paths into polygons and in the end smoothes and approximates them with a curve-fitter.

We have come across the following drawbacks of this service. First, VTracer does not work well with all image formats; for example, it produces a black background instead of a transparent one while processing

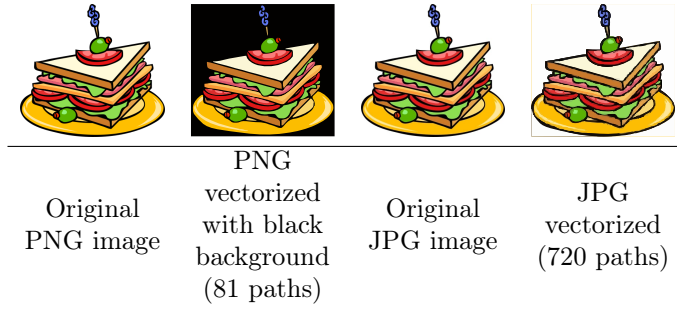


Figure 4. VTracer vectorization and its issues with the black background color and inaccurate vectorization of low-quality images.

PNGs, and there are no options to change this behaviour. Second, VTracer does not handle low-quality images well, creating many unnecessary and inaccurate shapes. In Fig. 4, we show an example of the black background appearance for a PNG high-quality image and the result for the same image converted to a low quality JPG.

§4. COMPARISON

4.1. Comparison Criteria. To make a valuable comparison of vectorization methods, it is necessary to consider that the visual appeal and similarity of the resulting vector image to the original raster image are not the only important criteria. The speed of vectorization is also an important factor.

The main advantages of a vector image are its simplicity and a small number of shapes used. Although a vector image can contain various shapes (circles, rectangles, paths, etc.), vectorization methods tend to generate images using only paths. Paths themselves consist of segments (Bèzier curves, straight lines, etc.) and their number in each path should be low as well. This is necessary both for simpler post-processing by designers and faster image transfer to the user through the Internet with subsequent vector image rendering.

Thus, the main five criteria for evaluating vectorization methods are:

- (1) similarity to the original bitmap;






















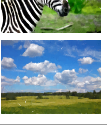
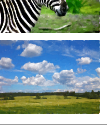
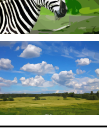
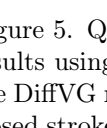
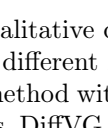
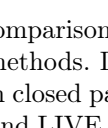
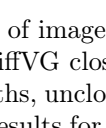
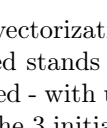
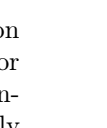
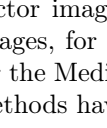
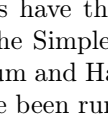
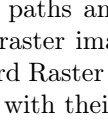
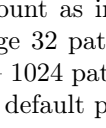
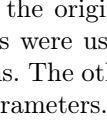
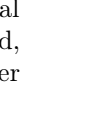
Image Description	Original	Mang2Vec	DVoTD	DiffVG (closed)	DiffVG (unclosed)	LIVE
Simple Vector						
Medium Vector						
Hard Vector						
Simple Raster						
Medium Raster						
Hard Raster						

Figure 5. Qualitative comparisons of image vectorization results using different methods. DiffVG closed stands for the DiffVG method with closed paths, unclosed - with unclosed strokes. DiffVG and LIVE results for the 3 initially vector images have the paths amount as in the original images, for the Simple raster image 32 paths were used, for the Medium and Hard Raster – 1024 paths. The other methods have been run with their default parameters.

- (2) the simplicity or complexity of the resulting image including the number of shapes and their parameters;
- (3) generation performance;
- (4) versatility, i.e., the ability to generate a fairly accurate copy of the input image without prior model training;
- (5) human control to adjust hyperparameters.

However, taking into account all criteria at the same time is challenging because the methods we consider have many different parameters that affect all criteria simultaneously. Typically, by changing one parameter one can achieve an increase in image processing speed but at the same time reduce the quality of the resulting image.

4.2. Experimental Setup. We selected 6 images for comparison: 3 rasterized vector images and 3 bitmaps of different complexity. The original target vector images before rasterization had the following number of paths: dragon had 25, burger – 62, red landscape – 100. Ideally, vectorization methods should create images consisting of an approximately similar number of paths in a short period of time. At the same time, it is hard to expect vectorization methods to account for every image detail on an original raster image, as an over-detailed vector image does not satisfy the simplicity criterion of having a relatively small number of paths. It is also desirable that simple monochrome patches should be decorated with a minimum number of shapes and the image subject should not be lost.

We compare the following methods (with publicly available implementations): Mang2Vec, Deep Vectorization of Technical Drawings (DVoTD), iterative DiffVG, and LIVE, together with online methods. The following models are not included in the comparison:

- (1) Im2Vec [23], because we could not confirm in practice the results described in the paper, and it also requires additional significant pre-training for processing relatively diverse images;
- (2) ClipGEN [27], because the model is limited to the set of predefined classes and there no publicly available implementation for it;
- (3) VAE and GAN introduced in DiffVG [17], because they also require additional pretraining; also, even on such a simple dataset as MNIST we found their results unsatisfactory;
- (4) algorithmic methods, since we found no implementations publicly available.

Figure 5 contains the original images and their vectorized versions using different methods reviewed in this work.

Our experiments have proven that the Mang2Vec and DVoTD models are not sufficiently versatile since they are capable of processing only black-and-white images. At first glance, Mang2Vec vectorizes the image well, but its significant drawback is the use of a very large number of shapes: for instance, the “burger” image on the last 5th iteration had 3065

paths and the “dragon” image on the 20th iteration had 16600 paths. Also, the method adds many `<clippath>` and `<circle>` tags, which seems useless. The method uses image splitting into patches and performs a separate vectorization of each patch, which is acceptable when processing detailed manga images. However, a large monochrome space becomes divided into a large number of shapes, which is unacceptable. In the Mang2Vec method, the user can specify a different number of iterations, but if one sets a small number, the boundaries of patches that the image is divided into become clearly visible. Since Mang2Vec automatically resizes input images to resolution 4096x4096, its working time is constant and equals 157 seconds.

Deep Vectorization of Technical Drawings (DVoTD) was meant to be able to use either quadratic Bézier curves or straight lines. We managed to run the method for curves, but the implementation of the second method (straight lines) is imperfect and the code is likely to contain some issues that lead to a crash during execution, which we could not fix. The method struggles to fill in contours with a solid color, as it was originally made to generate black stroke lines. We ran the method with default parameters and, for instance, the “dragon” image had 132 paths. The running time of the 270x480 image was 142 seconds, 373x405 – 179 seconds, 582x496 – 273 seconds.

Vectorization by the DiffVG iterative method can be done in two ways: generating images consisting of curves and of closed shapes. The approach is simple and quite effective, but it produces many artifacts with shapes that the method apparently attempts to hide, but it fails to succeed. In addition, the reconstruction of absolutely exact visual copies of the original vector images cannot be obtained even using a large number of paths (1024 shapes).

Different renderers convert a vector image to a bitmap in different ways. For example, images generated by DiffVG and LIVE will look inaccurate and careless when they are rendered by Inkscape. This behavior occurs due to the fact that these images contain curves protruding beyond the edges of the `viewBox` attribute, and Inkscape displays them instead of cropping them. At the same time, other renderers, for example, the one in the Google Chrome browser, process images correctly without extra curves that remain beyond the `viewBox`. However, these curves are still a problem, since they are superfluous and they create additional artifacts in the image code and unnecessarily enlarge it.

The LIVE method iteratively adds a layer consisting of one or more shapes specified by the user to the image and optimizes the resulting image. In addition, the number of image processing iterations after applying each layer should be specified manually. LIVE sets the number of iterations to 500 by default, but we noticed that after about 200 iterations, the image almost does not change, so we used this value in our experiments. In the optimization process, LIVE uses the DiffVG rasterizer to convert the current vector image into a raster image and compare it with the target image. Rasterization is performed by default at the same resolution as the target image, but for large resolutions it is computationally time-consuming. For example, for a resolution of 1080x1920, processing the first layer in 200 iterations on the Nvidia RTX 3090 Ti GPU took 212 seconds, then by the 10th layer processing of one layer reached 244 seconds. Finally, processing of 28 layers took almost 2 hours. Therefore, we decided to pre-scale raster images so that their maximum side does not exceed 512 pixels. At the same time, it is worth noting that this approach carries the risk of losing details in the image. With this approach, the processing of the first layer took 16 seconds, but by the 10th layer, the processing time of one layer was 40 seconds. The total processing time of 46 layers took about 32 minutes. When limiting the maximum image side to 256 pixels, the processing time of the first layer was 6 seconds, and overall 32 layers were processed in 22 minutes. We note that the processing time is also affected by the number of shapes in each layer. In our experiments we used less than 7 shapes in each layer only for the first 5 layers, after that we used 7, 10, 20 or 30 shapes per layer, gradually increasing the number.

LIVE creates the most accurate images among the ML methods. However, the most significant disadvantage of this method is a very long image processing time, which depends on the number of layer additions, the number of iterations of processing each layer, the dimensions of the image for which intermediate rasterization is performed.

It is worth noting that DiffVG also has problems with long intermediate rasterization; however, it is less noticeable due to the lower total number of iterations. The resulting DiffVG and LIVE processing times are shown in more detail in Table 1 and Table 2.

The requirement of DiffVG and LIVE models to directly control the number and type of applied shapes represents, on the one hand, their advantage, but on the other hand, since there are no good models for

Table 1. Running time of the DiffVG iterative algorithm at different startup parameters on an NVIDIA RTX 3090Ti GPU. Total iterations number is 500. There are no significant performance differences between methods with closed and unclosed paths.

Resolution	Total paths	Time (sec)
512x512	16	47
405x373	62	48
496x582	25	56
512x512	32	56
512x288	256	143
512x381	256	186
512x288	512	216
512x381	512	280
1920x1080	32	296
1920x1080	64	319
1920x1080	100	349
512x288	1024	389
1920x1080	256	435
512x381	1024	446
1920x1432	256	582
1920x1080	512	583
1920x1432	512	770
1920x1080	1024	837
1920x1432	1024	1099

determining the required number of shapes, automatic vectorization of a large set of varied raster images becomes almost impossible.

The online methods we have found show the best quality of image vectorization. However, this is achieved by using a large number of shapes, and their number can only be controlled indirectly by specifying the number of available colors. The results are presented in Table 3.

However, none of the considered methods could recreate exact copies using such a number of shapes.

Table 2. Running time of the LIVE algorithm with different startup parameters on an NVIDIA RTX 3090Ti GPU. Each layer is processed for 200 iterations.

Resolution	Layers schema	Layers	Paths	Time (s)
256x256	4x1	4	4	33
256x256	8x1	8	8	77
256x218	1,2,3,4,5x3	7	25	78
256x256	16x1	16	16	158
235x256	1,3,5x2,7x7	11	62	162
144x256	1,3,4,5,7,10x8	13	100	169
256x218	25x1	25	25	278
256x256	32x1	32	32	423
235x256	62x1	62	62	945
144x256	100x1	100	100	1231
471x512	1,3,5,7,10x24	28	256	1448
190x256	1,3,5,7,8,10x2,20x4,30x30	41	1024	2332
256x218	1,3,5,7,8,10x2,20x4,30x30	41	1024	2490
288x512	1,3,5,7,8,10x2,20x4,30x30	41	1024	3197
471x512	1,3,5,7,8,10x2,20x4,30x30	41	1024	3891
1080x1920	1,3,5,7,10x24	28	256	7150

Table 3. Classification and comparison of vectorization methods. 'ML' means machine learning, 'DL' means deep learning, and 'RL' means reinforcement learning.

Method	Approach	Code	Versatility	Speed	# of figures
DiffVG	ML: Iterative	+	+	Low	User-defined
Im2Vec	ML: VAE	+	-	Pretrain needed	User-defined
LIVE	ML: Iterative	+	+	Very low	User-defined
ClipGEN	ML: Iterative+DL	-	-	Pretrain needed	User-defined
Mang2Vec	ML: RL	+	-	Medium	Many
DVoTD	ML: DL	+	-	Medium	Many
VTracer	Algorithmic, online	+	+	Very High	Medium

§5. CONCLUSION

In this work, we have shown that current image vectorization methods are difficult to use in practice. Online methods without manual hyperparameter tuning create images containing a large number of paths, which

increases the amount of used memory, and design refinement becomes a time-consuming task. All existing machine learning-compatible methods also require human control and adjustment of the number of iterations for the method, output vector image parameters, etc. The Im2Vec model is not capable of storing and generating complex images and is not a universal vectorizer that could create a vector counterpart for any input image. The LIVE method is the only universal model that allows the user to control the number of drawn shapes; however, due to the use of an iterative approach, generating a single image takes a huge amount of time.

According to our measurements, DiffVG is the fastest among ML methods without much loss in quality. However, a large number of paths are required for high-quality results. At the same time, LIVE is able to get to roughly equivalent quality using fewer shapes. However, the main problem with the LIVE method is its overly long running time.

In general, it is perhaps best to use online methods, but they do not allow the user to adjust the number of applied shapes.

To summarize, vectorization methods still inevitably come with a trade-off between image quality, path number, segment number, closed or not paths are, number of iterations, and running time.

REFERENCES

1. A. Carlier, M. Danelljan, A. Alahi, and R. Timofte, *Deepsvg: A hierarchical generative network for vector graphics animation*, CoRR **abs/2007.11301** (2020).
2. W. Dai, T. Luo, and J. Shen, *Automatic image vectorization using superpixels and random walkers*, 2013 6th International Congress on Image and Signal Processing (CISP) **2** (2013), 922–926.
3. L. Deng, *The mnist database of handwritten digit images for machine learning research*, IEEE Signal Processing Magazine **29** (2012), no. 6, 141–142.
4. V. Efimova, A. Chebykin, I. Jarsky, E. Prosvirnin, and A. Filchenkov, *Neural style transfer for vector graphics*, 2023.
5. V. Efimova, I. Jarsky, I. Bizyaev, and A. Filchenkov, *Conditional vector graphics generation for music cover images*, arXiv preprint arXiv:2205.07301 (2022).
6. V. Egiазarian, O. Voynov, A. Artemov, D. Volkhonskiy, A. Safin, M. Taktasheva, D. Zorin, and E. Burnaev, *Deep vectorization of technical drawings*, Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII 16, Springer, 2020, pp. 582–598.
7. P. Esser, R. Rombach, and B. Ommer, *Taming transformers for high-resolution image synthesis*, Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2021, pp. 12873–12883.
8. K. Frans, L. B. Soros, and O. Witkowski, *Clipdraw: Exploring text-to-drawing synthesis through language-image encoders*, CoRR **abs/2106.14843** (2021).

9. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, Communications of the ACM **63** (2020), no. 11, 139–144.
10. G. J. Hettinga, J. Echevarria, and J. Kosinka, *Efficient image vectorisation using mesh colours*, The Eurographics Association, 2021.
11. A. Jain, A. Xie, and P. Abbeel, *Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models*, arXiv preprint arXiv:2211.11319 (2022).
12. S. Jeschke, D. Cline, and P. Wonka, *Estimating color and texture parameters for vector graphics*, Computer Graphics Forum, vol. 30, Wiley Online Library, 2011, pp. 523–532.
13. D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, arXiv preprint arXiv:1312.6114 (2013).
14. ———, *An introduction to variational autoencoders*, arXiv preprint arXiv:1906.02691 (2019).
15. Y.-K. Lai, S.-M. Hu, and R. R. Martin, *Automatic and topology-preserving gradient mesh generation for image vectorization*, ACM Transactions on Graphics (TOG) **28** (2009), no. 3, 1–8.
16. G. Lecot and B. Lévy, *Ardeco: automatic region detection and conversion*, Eurographics Symposium on Rendering, 2006.
17. T.-M. Li, M. Lukáč, G. Michaël, and J. Ragan-Kelley, *Differentiable vector graphics rasterization for editing and learning*, ACM Trans. Graph. (Proc. SIGGRAPH Asia) **39** (2020), no. 6, 193:1–193:15.
18. Z. Liao, H. Hoppe, D. Forsyth, and Y. Yu, *A subdivision-based representation for vector image editing*, IEEE transactions on visualization and computer graphics **18** (2012), no. 11, 1858–1867.
19. R. G. Lopes, D. Ha, D. Eck, and J. Shlens, *A learned representation for scalable vector graphics*, CoRR **abs/1904.02632** (2019).
20. X. Ma, Y. Zhou, X. Xu, B. Sun, V. Filev, N. Orlov, Y. Fu, and H. Shi, *Towards layer-wise image vectorization*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 16314–16323.
21. B. Price and W. Barrett, *Object-based vectorization for interactive image editing*, The Visual Computer **22** (2006), 661–670.
22. S. Pun and C. Tsang, *Vtracer*, 2020.
23. P. Reddy, M. Gharbi, M. Lukac, and N. J. Mitra, *Im2vec: Synthesizing vector graphics without vector supervision*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 7342–7351.
24. R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10684–10695.
25. C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. L. Denton, K. Ghasemipour, R. Gontijo Lopes, B. Karagol Ayan, T. Salimans, et al., *Photorealistic text-to-image diffusion models with deep language understanding*, Advances in Neural Information Processing Systems **35** (2022), 36479–36494.
26. P. Schaldenbrand, Z. Liu, and J. Oh, *Styleclipdraw: Coupling content and style in text-to-drawing translation*, arXiv preprint arXiv:2202.12362 (2022).

27. I.-C. Shen and B.-Y. Chen, *Clipgen: A deep generative model for clipart vectorization and synthesis*, IEEE Transactions on Visualization and Computer Graphics **28** (2021), no. 12, 4211–4224.
28. H. Su, J. Niu, X. Liu, J. Cui, and J. Wan, *Vectorization of raster manga by deep reinforcement learning*, arXiv preprint arXiv:2110.04830 (2021).
29. J. Sun, L. Liang, F. Wen, and H.-Y. Shum, *Image vectorization using optimized gradient meshes*, ACM Transactions on Graphics (TOG) **26** (2007), no. 3, 11–es.
30. S. Swaminarayan and L. Prasad, *Rapid automated polygonal image decomposition*, 35th IEEE Applied Imagery and Pattern Recognition Workshop (AIPR'06) (2006), 28–28.
31. X. Tian and T. Günther, *A survey of smooth vector graphics: Recent advances in representation, creation, rasterization and image vectorization*, IEEE Transactions on Visualization and Computer Graphics (2022).
32. Y. Wang and Z. Lian, *Deepvecfont: synthesizing high-quality vector fonts via dual-modality learning*, ACM Transactions on Graphics (TOG) **40** (2021), no. 6, 1–15.
33. G. Xie, X. Sun, X. Tong, and D. Nowrouzezahrai, *Hierarchical diffusion curves for accurate automatic image vectorization*, ACM Transactions on Graphics (TOG) **33** (2014), no. 6, 1–11.
34. M. Yang, H. Chao, C. Zhang, J. Guo, L. Yuan, and J. Sun, *Effective clipart image vectorization through direct optimization of bezigons*, IEEE Transactions on Visualization and Computer Graphics **22** (2016), 1063–1075.
35. J. Yu, Y. Xu, J. Y. Koh, T. Luong, G. Baid, Z. Wang, V. Vasudevan, A. Ku, Y. Yang, B. K. Ayan, et al., *Scaling autoregressive models for content-rich text-to-image generation*, arXiv preprint arXiv:2206.10789 (2022).
36. R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, *The unreasonable effectiveness of deep features as a perceptual metric*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 586–595.
37. S. Zhao, F. Durand, and C. Zheng, *Inverse diffusion curves using shape optimization*, IEEE transactions on visualization and computer graphics **24** (2017), no. 7, 2153–2166.
38. H. Zhou, J. Zheng, and L. Wei, *Representing images using curvilinear feature driven subdivision surfaces*, IEEE transactions on image processing **23** (2014), no. 8, 3268–3280.

ITMO University

E-mail: ivanjarsky@niuitmo.ru

E-mail: ivanjarsky@niuitmo.ru

E-mail: vefimova@itmo.ru

GO AI LAB

E-mail: aaafil@gmail.com