上海交通大学学位论文

基于深度学习的图像矢量化

姓　　名：JUANG IAN CHONG

学　　号：51903990028

导　　师：易冉

学　　院：电子信息与电气工程学院

学科/专业名称：计算机科学与技术

申请学位层次： 学士

2023 年 5 月

A Dissertation Submitted to

Shanghai Jiao Tong University for Bachelor Degree

RASTER IMAGE VECTORIZATION WITH DEEP LEARNING

Author: JUANG IAN CHONG

Supervisor: RAN YI

School of Electronic Information and Electronic Engineering

Shanghai Jiao Tong University

Shanghai, P.R.China

June 28th, 2023

上海交通大学

学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全知晓本声明的法律后果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

上海交通大学

学位论文使用授权书

本人同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。

本学位论文属于 ：

□**公开论文**

□**内部论文**，保密□1 年/□2 年/□3 年，过保密期后适用本授权书。

□**秘密论文**，保密____年（不超过 10 年），过保密期后适用本授权书。

□**机密论文**，保密____年（不超过 20 年），过保密期后适用本授权书。

（请在以上方框内选择打“ √ ”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘 要

一个图像的矢量化表示应该以最小的拓扑复杂度代表原始图像。通过传统方法生成的矢量图形通常过于复杂。近来的发展重点研究基于深度学习或基于优化的方法来生成矢量图形。深度生成模型可以生成具有代表性拓扑结构的矢量图形，但它们仅限于训练数据的领域。相对而来，直接优化算法能在任意图像领域上取得很好的效果。 然而，当前最先进的图像矢量化优化算法要么在矢量质量方面低于标准，要么非常耗时才能获得高质量的矢量图形，原因在于优化算法中的形状初始化是一个难以解决的问题。

通过图像分割，我们可以直接生成适合优化的初始形状。在此论文，我们提出一个多阶段图像矢量化框架，并赋予其名为 SAMVG。SAMVG 利用最新的任意图像分割模型 Segment-Anything Model (SAM) 来进行图像分割，并利用可微分渲染化来进行直接优化。 通过一系列广泛的实验，我们验证 SAMVG 可以在任何领域生成高质量的 SVG，同时与以前的最新技术相比需要更少的计算时间与资源。

关键词：图像矢量化、矢量图形、计算机图形学、计算机视觉

ABSTRACT

A good vector representation of an image should be highly representative of the original image with minimum topological complexity. Vector graphics generated via traditional methods are often redundantly complex. Recent works focus on deep learning-based or optimization-based methods to generate vector graphics. While deep generative models can produce vector graphics with representative topology, they are limited to the domain of the training data. Direct optimization algorithms can achieve good results generally. However, current state-of-the-art optimization algorithms for image vectorization are either sub-par in terms of vector quality or extremely time-consuming to achieve high quality vector graphics, because a good initialization is critical yet non-trivial for direct optimization for image vectorization.

Using image segmentation, we can produce good initial shapes to undergo direct optimization. We introduce SAMVG, a multi-stage framework to vectorize raster images into SVG utilizing general image segmentation with the Segment-Anything Model and direct optimization with differentiable rendering. Through a series of extensive experiments, we demonstrate that SAMVG can produce high quality SVGs in any domain while requiring less computation time and complexity compared to previous state-of-the-arts.

Key words: image vectorization, vector graphics, computer graphics, computer vision

CONTENTS

Chapter One Introduction

1.1 Foreword

In computer graphics, images are either represented by a dense grid of colored pixels (raster images) or a collection of parametric shape primitives (vector graphics)[4]. While most images are represented in raster form due to their versatility and rich details, vector graphics have their own advantages. Vector graphics are particularly useful when it comes to creating graphics that need to be scaled up or down without losing quality, such as logos and icons. The reason vector graphics are so compact is that they store images as a series of mathematical equations rather than individual pixels. This makes them ideal for situations where file size is a concern, such as when creating graphics for websites or mobile applications.

Compared to rendering raster images which is now a matured technique, automatically generating vector graphics remains a significant challenge, especially for more complex images such as photos or artworks. Despite this, there already exist numerous works[2, 3, 5, 6] that generate high-quality vectors that are nearly visually identical to their raster counterpart. However, vector graphics generated from these methods are always too complicated, as in, having too many parameters than necessary for editing and do not preserve key topological features of the original image, which is a significant advantage of vector representation. For example, [3]represent vector graphics in meshes, which consist of many paths that have virtually no impact on the shape topology of the image. Recent advances in deep learning and differentiable rendering methods inspired various works[7-13] to tackle image vectorization via deep learning generative models or direct optimization. These new methods aim to generate vector graphics that are not only visually similar to their raster counterparts but also preserve the topological features of the original image, making them easier to edit and work with. Nevertheless, both of these methods have their own drawbacks:

Deep generative models[9-14] are domain-specific to their training data, and high-quality SVG training data is difficult to obtain at the scale that deep learning usually requires. This

limitation can make it challenging to generate high-quality vector graphics that are representative of a broad range of images. Some recent works [13, 14] have attempted to address this issue by utilizing differentiable rendering[8] to perform supervised training with respect to raster images. However, the rendering process is computationally heavy, resulting in inefficient training, and the representative capability of the model is still limited to extremely simple vector graphics.

Direct optimization algorithms[7, 8, 15] are highly generalizable, but they are heavily reliant on a good initialization to achieve good results. One recent study, LIVE[7] addresses this by adding only a few shape primitives (one by default) at a time, optimizing to convergence each iteration. Unfortunately, this approach can be extremely computationally expensive, making it impractical for batch processing to create SVG datasets for deep learning models.

1.2 The main content of this paper

In this paper, we attempt to address the initialization problem encountered in direct optimization-based image vectorization algorithms in order to obtain high-quality image vectorization with a reasonable level of efficiency. As is well-known, one popular method for vectorizing images involves segmenting them into their constituent color components, as has been demonstrated in previous research[16, 17]. However, this approach becomes increasingly challenging as the complexity of the image increases, making it difficult to identify the components that accurately represent the underlying semantics of the original image without introducing a significant amount of topological complexity into the model. Therefore, our proposed approach is designed to address this issue by utilizing advanced segmentation techniques that are capable of segmenting the whole image semantically.

Recently, Kirillov et al.[18] proposed a new task coined *Segment-Anything*, where the objective of the task is to generate a valid segmentation mask for any object in any image given a prompt proposal. To achieve this, they developed a large-scale model called the Segment-Anything Model (SAM), which was trained on a massive dataset comprising over a billion segmentation masks. SAM is capable of generating segmentation masks for any part of a generic image based on a prompt input, making it ideal for segmenting images for

path tracing purposes. Building on this work, we developed a novel multi-stage framework that aims to convert raster images into high-quality SVGs with reasonable runtime, which we refer to as SAMVG. The SAMVG framework involves using SAM to generate segmentation masks for the entire image, which are then filtered using a novel method. We then prompt SAM to generate masks for the regions that were missed in the first pass. Using the set of masks produced by SAM, we trace each component of these masks into Bèzigons to generate an initial SVG. We then further optimize the SVG generated using differentiable rendering[8] with respect to the target image. Lastly, we identify those regions that require more shape primitives to be represented via convolution on the error map. The centers of the regions identified are input as prompts to SAM to retrieve another set of masks to add to the SVG before performing one final round of optimization.

In our ongoing efforts to further improve the performance of our proposed framework, SAMVG, we implemented several variations of the original model to investigate the impact of different factors on its overall efficacy. Specifically, we explored the use of a perception-based loss for optimization, the incorporation of learnable alpha values for translucent shapes, and the utilization of a variable number of segments for each shape. To evaluate the effectiveness of these variations, we conducted experiments using two different datasets: a self-collected diverse dataset comprising images from various domains to test the generalization ability of SAMVG, and a classification dataset to assess the representation ability of SAMVG with a deep classification model. We quantitatively evaluated the results of these experiments against current state-of-the-art techniques as baselines and found that SAMVG outperformed previous methods across multiple image evaluation metrics while being significantly more efficient in terms of run time. To further demonstrate the generalization ability of SAMVG, we examined its performance for each category in the self-collected dataset and found that the superiority of SAMVG is consistent across the majority of the categories. Moreover, we compared the outputs produced by SAMVG with those of other methods qualitatively and concluded that SAMVG is able to generate shapes that are more in line with the semantic information of the target image. However, we recognize in the qualitative study that there are still certain limitations to the current implementation of SAMVG, as evidenced by some of the artifacts present in the vector graphics produced. As such, we have also discussed potential areas of improvement for

future research. Overall, our findings suggest that SAMVG is a highly effective and efficient framework for generating high-quality vector graphics from raster images, with minimal number of SVG parameters thus more suitable for human editing and as training data for machine learning.

1.3 The significance of this article

The development of our proposed framework, SAMVG, represents a significant contribution to the field of image vectorization and has important practical implications. Specifically, SAMVG is designed to generate high-quality vector graphics from raster images that are highly representative of their original topological features while maintaining simplicity. We believe that this approach has significant practical value, as previous methods for producing SVGs with good topological representations, such as LIVE[7], can take hours to vectorize a high-resolution image. In contrast, SAMVG can generate SVGs with superior topology within minutes, thanks to the initial vector produced by SAM, which better correlates with the semantic information of the target image. Moreover, the development of SAMVG also has important implications for the generation of large-scale datasets in SVG format, which is crucial for the advancement of deep learning methods in the SVG domain.

SAMVG represents one of the earliest downstream applications of the Segment-Anything Model (SAM), which was trained on a massive dataset comprising over a billion segmentation masks. As such, our work provides valuable insights into effective prompt engineering and proposes a novel filtering method to obtain the best masks for downstream tasks. These insights and methods are not only applicable to image vectorization but can also be used for other downstream tasks based on SAM, such as semantic segmentation or object detection.

Furthermore, our experimental results demonstrate that the quality of the vectors generated by SAMVG is on par with, if not better than, the current state-of-the-art methods. Through a qualitative analysis of the results, we have also identified certain limitations of optimization-based image vectorization algorithms, providing valuable insights for future research in the field.

Overall, our work represents an important step forward in the development of more

effective and efficient methods for generating high-quality semantically aware vector graphics from raster images, with significant practical applications and implications for future research in image analysis and computer vision.

1.4 Summary

In summary, our paper makes several important contributions to the field of image vectorization. Firstly, we introduce a novel pipeline for generating high-quality vector graphics from raster images that are highly representative of the original image in terms of human perception, while maintaining high efficiency. This approach opens up new avenues for future research in the SVG domain, with significant practical applications in a variety of domains.

Secondly, we design custom prompts and filtering methods to identify the best dense segmentation map for the entire image, which can be used not only for image vectorization but also for other downstream applications of the Segment-Anything Model (SAM). These methods represent an important contribution to the field, with potential implications for a wide range of applications in image analysis and computer vision.

Finally, our experimental results demonstrate that SAMVG is capable of achieving state-of-the-art quality in image vectorization with a fraction of the run time needed by existing methods such as LIVE. Through a comprehensive analysis of the experiment results, we also identify certain limitations of optimization-based image vectorization algorithms and outline potential areas for improvement and future research directions.

Overall, our paper represents a significant step forward in the development of more effective and efficient methods for generating high-quality vector graphics from raster images, with important practical applications and implications for future research in image analysis and computer vision.

Chapter Two Related Work

2.1 Image vectorization

2.1.1 Traditional Image Vectorization Algorithms

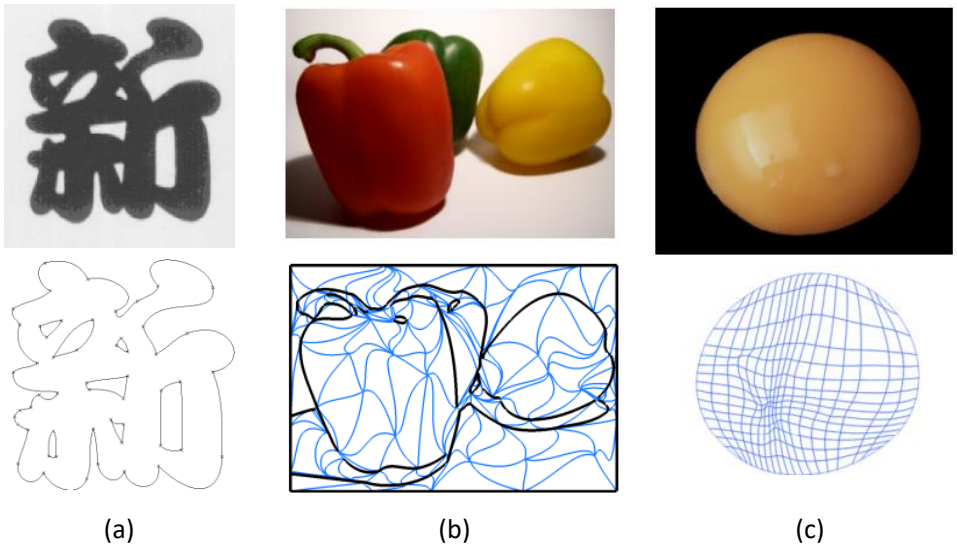

Figure 2-1: Illustration of the vectorization results and topology of representative traditional vectorization algorithms. (a): The classic font tracing algorithm[1]. (b): Patched-based image vectorization[2]. (c): Image vectorization with gradient meshes[3]. Images are retrieved from each respective literature.

Image vectorization has been a well-researched task in the field of computer graphics for several decades. The earliest studies on image vectorization can be traced back to generating fonts from scanned text documents[1, 19]. where the process involved selecting corner points from an edge map and using the least squares method to fit bézier curves into a closed path. Hoshyari et al.[20] built on top of this paradigm and proposed a more advanced boundary vectorization method more in line with human perception. As demand for vectorizing color images increased, various studies[2, 21-23] focused on applying different segmentation techniques to obtain edge maps for curve fitting. In[2], a raster image is

segmented into a composite of small bèzior patches, giving a high-quality but dense and complicated vectorization. In[21] and[23], the image vectorization process is semi-automatic, where the user first provides an initial instruction, a prompt in the form of a point or area selection to segment the image based on tree search exploration algorithms. In[22], B. Kim et al. used semantic segmentation with neural networks before vectorization, which is most similar to our work. However, the scope of their research is only within the domain of line drawings, while our work attempts to vectorize any raster images, including photographic images. There are also numerous commercial tools[17, 24, 25] available online that follow the segment-and-fit method to perform image vectorization.

Another line of research in image vectorization has focused on methods that are independent of segmentation, including diffusion curves[6, 16, 26] and gradient meshes[3]. These methods can generate shape primitives with gradient color to produce vector graphics that are highly detailed, rich in color variations and visually almost identical to the target image. However, these methods have some disadvantages. Firstly, the generated vectors can be highly complex, making them less efficient and more difficult to work with. Additionally, the vectors produced by these methods are often less representative of the semantic geometrical information of the target image, which can limit their usefulness for downstream applications in image analysis and computer vision.

2.1.2 Vectorization with differentiable rendering

With the increasing popularity of deep machine learning, there has been a shift in research interest towards optimization-based image vectorization methods. As classical rasterization algorithms for bézier curves are not differentiable[27-29], differentiable loss functions are required for either direct optimization or deep learning methods supervised by raster images. In response to this challenge, Yang et al.[15] found a differentiable approximation function to render closed bèzigons and combined this rendering method with customized heuristics to perform direct optimization on disjoint bèzigons. Li et al.[8] proposed a more general differentiable rasterizer that supports occlusion, transparency, rational polynomials, and strokes. This work paved the way for a multitude of future research regarding optimization and deep learning in the SVG domain.

Using differentiable rendering, Ma et al.[7] proposed a layer-wise method, coined LIVE, to vectorize images by iteratively adding shape primitives to the image while optimizing for each shape added. This approach has been shown to produce high-quality vector graphics that are highly representative of the original image. Another popular approach that utilizes differentiable rendering is to train deep learning models to vectorize images. Various deep learning methods were explored for vectorizing images, including Recurrent Neural Networks (RNN)[13], Long Short-Term Memory Networks (LSTM)[12], and Deep Reinforcement Learning methods[14]. In[13], Reddy et al. used an encoder-decoder architecture with convolutional neural networks (CNN) and RNNs to extract bèzior curve parameters from latent space, which was trained in a self-supervised fashion from the input raster images. In[12], Shen et al. proposed a two-stage pipeline with CNN and LSTM to synthesize clip art from raster images with additional loss functions to regularize the geometry of generated shapes. In[14], Su et al. introduced reinforcement learning methods to vectorize Japanese Manga comics using stroke primitives.

An important consideration in image vectorization is the need to represent the key topological features of an image semantically, similar to how a human would construct a minimum number of shapes with the simplest topology that can represent the target image. Therefore, it is crucial for a model to understand the underlying structure of the image perception-wise Chan et al.[30, 31], combined the differentiable rasterizer with CLIP loss[32] to propose two optimization-based methods to generate semantically aware line drawings for objects[30] and scenes[31].

2.2 Deep learning with vector graphics

Other than plain image vectorization, the rise of deep learning technologies also sparked research interest in deep learning with vector graphics. For example, Lopes et al. [11]encoded SVG into latent space using variational autoencoders (VAE) to perform reconstruction, font generation and interpolation between two vector graphics. Similarly,[9, 33] utilized hierarchical transformers as encoders and decoders for SVG manipulation, where [9] is implemented for general SVG manipulation like[11], while [33] focused on Chinese font generation. Jan, Ajay et al. [34]proposed a text-to-SVG generation model using diffusion

models and vectorization techniques from LIVE[7] with additional CLIP guidance. Efimova, V et al.[35] generated vector graphics of album covers from corresponding musical tracks using generational adversarial networks (GAN) with differentiable rendering for adversarial learning. Later, they also propose a neural style transfer method in vector domain using direct optimization with GAN-based content loss and LPIPS-based style loss[36].

2.3 Segment-Anything

The core idea of our work is to combine segmentation with optimization for image vectorization. A straightforward approach would be to use color-based segmentations like[17, 21, 25]. However, these types of methods often create overly complicated segmentations for complex photographic images and do not necessarily reflect the key semantics of the image. Another approach would be to use large-scale deep semantic segmentation models[35, 37, 38], which can capture key semantically representative shapes. Yet, these models are domain-specific and only capture specific objects that the model was trained for. Therefore, they are not suitable for a generic vectorization algorithm.

Figure 2-2: Examples of "valid" segmentations given a point prompt provided by the Segment-Anything Model (SAM). The source image is taken from online.

To address these limitations, we use the Segment-Anything Model (SAM)[18] which is trained on a large-scale dataset of over one billion masks and 11 million images. Rather than segmenting specific labeled objects, SAM is tasked with generating any reasonable segmentation given a prompt that can include labeled points, bounding boxes, masks, or texts[1]. This approach increases the generalization ability of SAM and makes it suitable for a

[1] While presented in the SAM paper, the current released version of SAM does not support text prompts.

wide range of machine learning tasks under the image recognition topic. The authors of SAM intended SAM to be the foundation model for most if not all machine learning tasks under the image recognition topic, and implemented several prototype versions of zero-shot down-stream applications of SAM, including image classification, object detection, object segmentation, edge detection and text-to-mask segmentation. We find that SAM is exceptionally suitable for our use case due to its powerful generalizable segmentation ability, and use SAM as a fundamental backbone of our framework.

2.3 Summary

In this chapter, we introduced the related work in the field of image vectorization, including traditional image vectorization methods that can be categorized into segmentation-based or non-segmentation-based. The advancement of these algorithms allowed for extremely high-quality vectorization. However, they do not represent the key topology information of the raster image well. We also introduced optimization-based and deep learning model-based image vectorization algorithms enabled by differentiable rendering, which each have their individual drawbacks. Moreover, we discuss relevant studies focused on SVG representation with deep learning. Finally, we introduce the foundation work of our work, the Segment-Anything Model which enabled universal semantic segmentation for any image.

The overall pipeline of our framework, SAMVG, is a union of traditional vectorization algorithms using segmentations and direct optimization algorithms using differentiable rendering. Specifically, SAMVG consists of 3 main components of segment, trace and optimize. Each component of SAMVG is inspired by different lines of work in the image vectorization domain The “segment” component utilizes the latest Segment-Anything technology, the “trace” component is based on the earliest classic font tracing algorithm and the “optimize” component is made possible by the emergence of differentiable rendering. Unlike traditional algorithms that use color-based segmentation, our pipeline uses a deep segmentation model that can better represent the semantic information of the image. This approach ensures that the generated vectors are highly representative of the original topological features of the image. Furthermore, unlike other direct optimization algorithms

such as LIVE, SAMVG has a much better starting point to optimize from, leading to faster vectorization speed and better vectorization quality, as demonstrated in our experiments.

Chapter Three SAMVG

In this chapter, we present the framework of SAMVG in a top-down fashion, where we start with the high-level framework of the stages in SAMVG, and then describe each component in detail in the following sections.

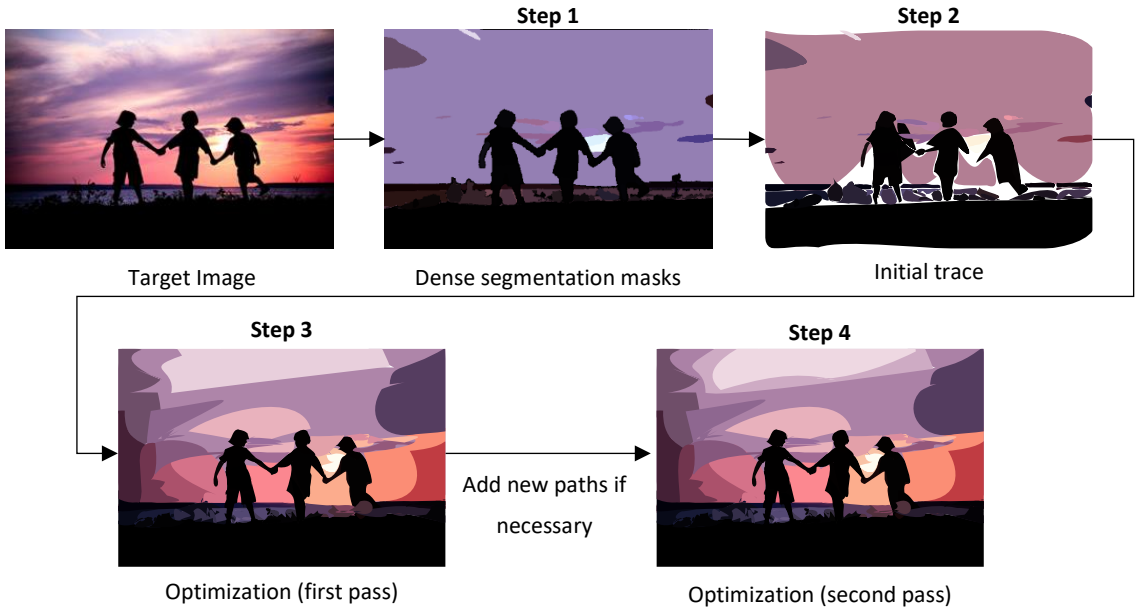

Figure 3-1: High-level overview of SAMVG. Illustration showcases the intermediate results of each step. Best viewed in color.
Step 1: the combined segmentation masks retrieved.
Step 2: The initial approximate trace of each mask.
Step 3: The SVG after first pass of optimization
Step 4: The final refined SVG.
Any missing components in step 3 would be added in Step 4.

3.1 High-level framework

SAMVG is a multi-stage framework consisting of segmenting, filtering, tracing, and optimizing. This approach can be separated into four main steps, each of which plays a

crucial role in the overall image vectorization process. We briefly describe each step as follows:

1. Given a target image $\boldsymbol{I} \in \mathbb{R}^{w \times h \times 3}$, generate a list of segmentation masks $m_1, \ldots, m_n \in \mathbb{R}^{w \times h}$ using SAM. To further enhance the segmentation quality, centers of missing components are located to be used as prompts to SAM for a second time. Redundant masks are filtered out, and the remaining masks are sorted by area to prepare for the next step.
2. For each component in the masks, the approximate shapes of the mask are traced with Bèzier curves to produce an initial SVG $\boldsymbol{S}$. This step is crucial for obtaining an accurate initial SVG that can be further optimized in the following steps.
3. We denote the rasterized image of $\boldsymbol{S}$ as $\boldsymbol{I}' = R(\boldsymbol{S})$ for a set number of iterations by computing mean square error (MSE) loss with target image $\boldsymbol{I}$ using differentiable rendering. Additional regulating losses are also used to improve the smoothness of the shapes and reduce artifacts.
4. Convolution with a circular kernel is performed on the difference map between $\boldsymbol{I}'$ and $\boldsymbol{I}$ to detect missing components. If such missing components exist, they are added to $\boldsymbol{S}$ by repeating step 1 and 2. The resulting SVG is optimized again for a final number of iterations to produce the final result.

The intermediate results of each step are illustrated in Fig. 3-1, which provides a visual representation of the image vectorization process in SAMVG. The details of each step are covered in the following sections, which provide a comprehensive overview of the SAMVG framework and its various components.

3.2 Retrieving segmentation masks

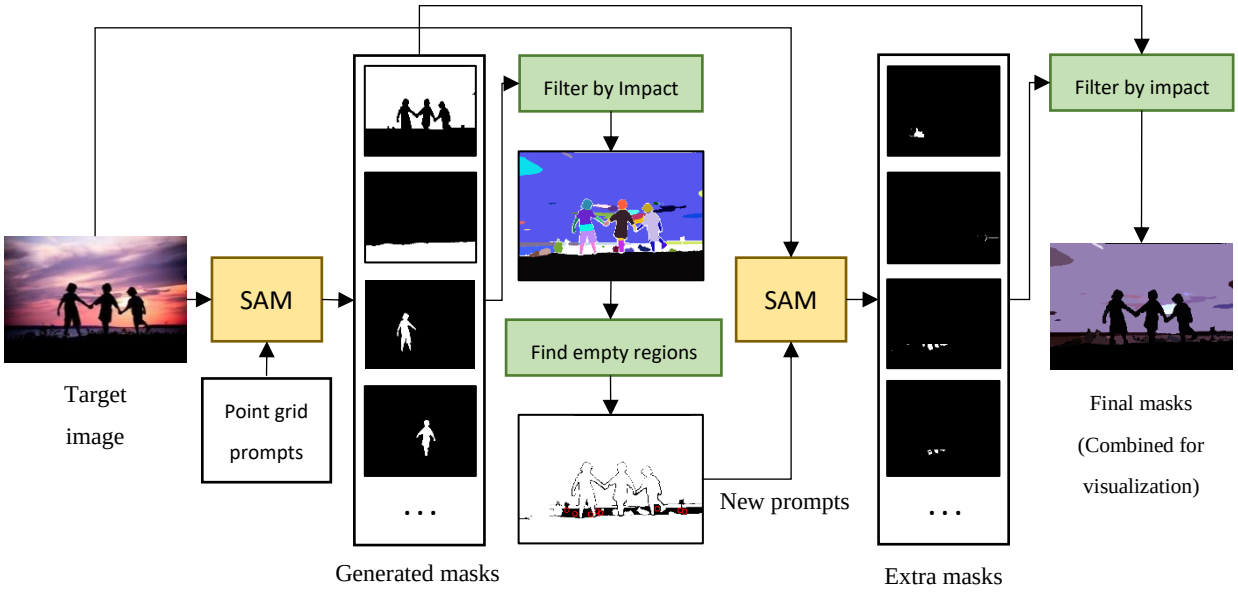

Figure 3-2: Flow diagram of the first stage of SAMVG to retrieve high quality segmentation masks. In addition to the filtering by SAM, we further filter the masks by testing its impact on the image render, and prompt the model twice for any components missed.

The first “segment” stage of SAMVG utilizes the Automatic Masks Generator (AMG) provided by SAM. AMG prompts the model with a regular grid of 32×32 points, which can be customized to adjust the density of the grid. Additionally, image crops are included to ensure that the model can capture segmentation details in different parts of the image. The AMG outputs a list of segmentation masks, which are then filtered by the model's predicted confidence score and intersection over union (IoU). Finally, the masks are post-processed to remove small components and holes, enabling easier path tracing in the later stages.

While the list of masks generated by AMG provides a good starting point, it is often insufficient to move on to the tracing step immediately. Depending on the filter threshold value and the complexity of the image, AMG may predict either too many redundant masks or too few masks, leaving large regions of the image uncovered, as shown in Fig 3-3. To address this issue, we introduce a simple filtering method called "Filter by Impact" on top of the existing filtering methods used to ensure that all kept masks are correct and significant to the image. Furthermore, we use convolution with a circular-sized kernel to identify

candidate center points of large uncovered regions, which are then used as prompts to SAM to generate additional segmentations.

In the following sub-sections, we provide detailed explanations of the filtering masks and the process of finding empty regions for prompts.

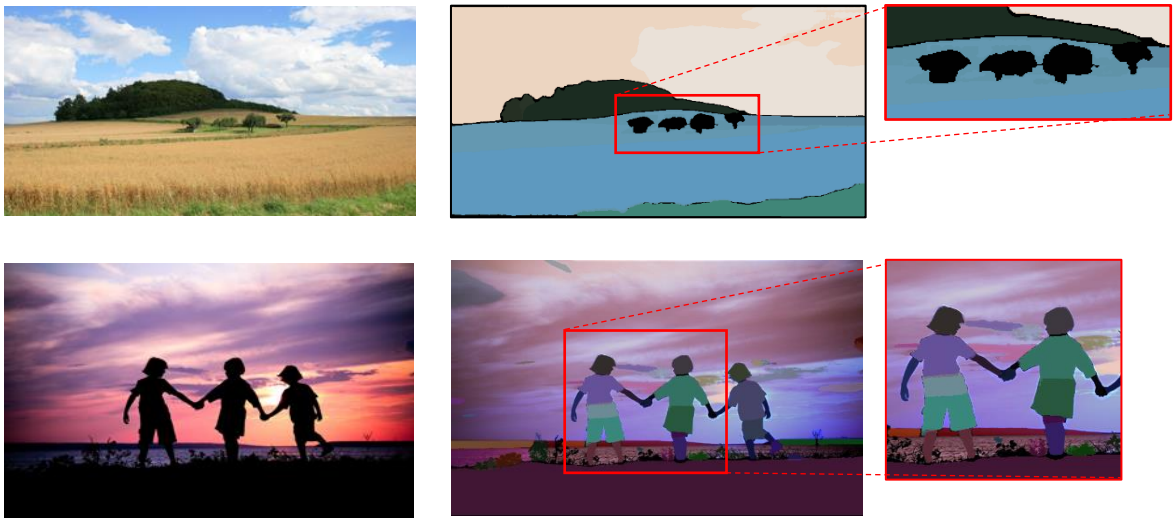

Figure 3-3: Examples of failure cases from the Automatic Masks Generator provided in SAM. Top row: missing masks for small components. Bottom row: unnecessary masks, these masks on the children are technically "sensible", however is not significant under this context and should be filtered *out*.

3.2.1 Filter by impact

To filter out undesirable masks, we evaluate the impact of the masks after rendering it on a canvas. Specifically, given a target image $\boldsymbol{I} \in \mathbb{R}^{w\times h\times 3}$, we start with a blank canvas $\boldsymbol{C_0} \in \mathbb{R}^{w\times h\times 3}$. We sort masks $m_1, \dots, m_n \in \{0,1\}^{w\times h}$ by area in descending order and assign color $c_i \in \mathbb{R}^3$ to m_i by averaging the regions covered by m_i on $\boldsymbol{I}$. We add mask m_i to the canvas sequentially:

$$\boldsymbol{C}_i = f(\boldsymbol{C}_{i-1}, m_i, c_i)\,, \tag{3-1}$$

where $f\colon \mathbb{R}^{w\times h\times 3} \to \mathbb{R}^{w\times h\times 3}$ sets pixels covered by m_i as c_i and calculates the normalized mean square error.

$$e_i = \frac{||\boldsymbol{I} - \boldsymbol{C_i}||_2}{Mask\ Area} \ . \tag{3-2}$$

Note that pixels that are not covered by any masks on $\boldsymbol{C}$ are set to have maximum error to avoid biases towards bright pixels. We define the impact γ_i of m_i as the difference between the new error and the previous error $\gamma_i = e_i - e_{i-1}$. We discard m_i if γ_i is lesser than a set threshold and revert $\boldsymbol{C_i}$ to the previous state $\boldsymbol{C_{i-1}}$. We visualize this process in Fig 3-4. Effectively, we are judging whether adding the current mask to the canvas brings the render closer to the target image.

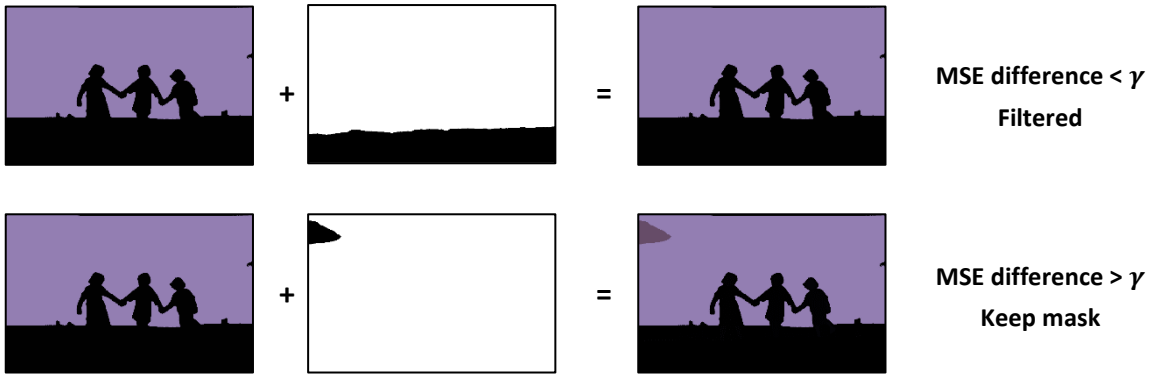

Figure 3-4: Illustration of filtering by impact. Top row: adding the ground mask makes no significant improvements to the mean square error of the image, therefore the mask is filtered. Bottom row: adding the cloud mask improves the mean square error of the image by more than a set threshold, therefore the mask is kept.

The "Filter by Impact" method used in SAMVG has several advantages. It retains masks that represent a sub-part of an object that may have a large intersection over union (IoU) with the mask of the parent object while filtering out small meaningless masks or incorrect masks that do not represent any object in the image. This approach ensures that all generated masks are significant and contribute to the overall representation of the image. The filtering process is applied every time SAM is prompted for new masks throughout the pipeline process to ensure that the list of generated masks contains only the most relevant and significant masks, reducing redundancy and increasing the efficiency of the vectorization process.

3.2.2 Locating empty regions

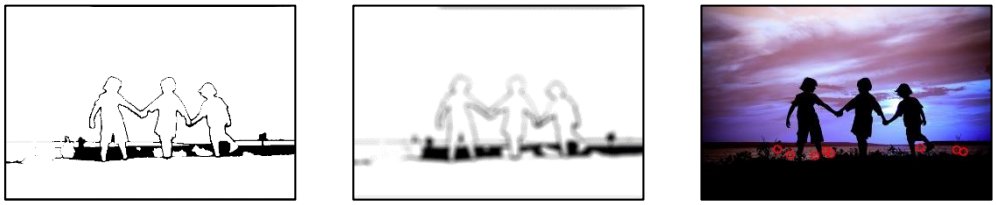

Figure 3-5: Visualization of the process for getting prompts for empty regions.

To identify large empty regions in $\boldsymbol{C}$, we first generate an alpha mask $\boldsymbol{C}_\alpha \in \mathbb{R}^{w \times h} = \bigvee_{i=1}^{n} m_i$, where $\vee$ stands for the bitwise-OR-operator. Subsequently, convolution is performed on $\boldsymbol{C}_\alpha$ with a fixed circular kernel $k \in \{0,1\}^{r \times r}$, where

$$k_{i,j} = \begin{cases} 1 & \text{, if } \sqrt{\left(i - \frac{r}{2}\right)^2 + \left(i - \frac{r}{2}\right)^2} \leq r \\ 0 & \text{,otherwise} \end{cases} . \tag{3-3}$$

The remaining coordinates that have values of zero in $\boldsymbol{C}_\alpha$ are selected as candidate points. The convolution, as we show in Fig. 3-5, has the effect of filtering out long gaps between the masks and selecting center of large regions that are completely empty within a circle of radius r. The radius is selected based on a fraction of the image size. Finally, we perform mean shift clustering[39] on the candidate points to generate points as prompts for SAM. After filtering, we arrive at a final list of masks $m_1, \ldots, m_n$ that are ready to be traced.

3.3 Approximate tracing

The second stage of SAMVG is the "trace" component, which involves generating a vector path from the set of generated masks. While there are many advanced tracing algorithms available in the literature[2, 17, 20, 24], we chose a modified version of the basic curve fitting algorithm to limit the complexity of the shape in order to improve optimization efficiency. This is because the shape primitives are optimized in later stages, allowing for some error in the tracing process. Our algorithm is similar to those used in previous studies,

such as[1, 19], the biggest difference of our tracing method lies in our method of selecting the corner points (i.e., endpoints of each bèzior curve in the path). For each contour point $p_i \in \mathbb{N}^2$ in the edge map $p_0, \dots, p_n$ generated from the mask, we assign a score s_i to p_i to estimate how its likelihood of being a corner point. The score is calculated by approximating the curvature of p_i as the ratio of the dot product and the distance product of the vectors of p_i from its k-neighbours

$$s_i = \frac{(p_i - p_{i+k}) \cdot (p_i - p_{i-k})}{|p_i - p_{i+k}| \cdot |p_i - p_{i-k}|}, \tag{3-4}$$

where $k = \alpha n, \alpha \in [0,1]$ is a fraction of the number of contour points in each mask. Note that the values $i + k$ or $i - k$ are more accurately $(i + k) mod\ n$ and $(i - k) mod\ n$ to handle out of bounds values, and the expression in Eqn. 3-4 is simplified for readability. Up to this point, the corner-finding procedure is exactly the same as the algorithms from the literature. Different from the literature, which selects corner points based on local maximas, we select the global maxima as the first corner point, then discard nearby points of the selected corner point from the list of potential candidates for corner points. We then search for the next corner point in the remaining candidates. The process is repeated until a set number of corner points is reached. The motive behind the modification is to fix the number of segments in each path for simplicity and comparability with other baselines with a fixed number of segments. In our experiments, we also implement the non-modified version with a variable number of segments as a variation to study its effects.

Using the corner points as endpoints, we can fit multiple bèzior curves to the edge map using the non-linear least squares method[40]. The first fitting process assumes a uniform distribution for the parameter values of each point. i.e., given point $p_1, \dots, p_n$ to be fitted to a bèzior curve $p = B(t)$, we set the t_i parameter that corresponds to p_i to be i/n. However, this may not be the optimal parameter value for the desired bèzior curve. Therefore, if the error value of the fitted curve is larger than a threshold, we reparametrize t_i using the following formula:

$$[B(t_i) - p_i] \cdot B'(t_i) = 0\ , \tag{3-5}$$

which can be approximated using Newton-Raphson's method. The new t_i value is used as the parameter for p_i to be fitted again. In the literature, the curves are checked for error again, after the second phase of curve fitting. If the error in the resulting bèzior curve is still larger than the threshold, the sampled points are split into two segments to be fitted with two bèzior curves. The splitting process repeats recursively until the error is below the threshold. We skip this step for efficiency and to fixate on the total number of segments in each path.

3.4 Optimization

After the tracing stage, an initial traced SVG is generated, which serves as an excellent starting point for the optimization process. Using differentiable rendering[8], we are able to perform gradient descent on the SVG parameters, which include point coordinates and fill color. The main loss function used for gradient descent is the mean square error between the render and the target image as the main loss function. In addition to the mean square error loss, we utilize a shape-regulating loss named Xing loss introduced by Ma et al[7], which penalizes shapes that are prone to self-interaction and encourages the shapes generated to be topologically sound. The idea behind Xing Loss is to add a constraint on the control points. Given 4 control points of a cubic bèzior curve denoted A, B, C and D, Xing Loss encourages the angle between $\overrightarrow{AB}$ and $\overrightarrow{CD}$ to be greater than $180°$. This is done by calculating two values D_1 and D_2:

$$D_1 = \begin{cases} 1, & x < 0 \\ 0, & x \geq 0 \end{cases}, \quad x = \overrightarrow{AB} \times \overrightarrow{CD} \ , \tag{3-6}$$

$$D_2 = \frac{\overrightarrow{AB} \times \overrightarrow{CD}}{||\overrightarrow{AB}|| \cdot ||\overrightarrow{CD}||}. \tag{3-7}$$

D_1 represent a conditional flag that is true when if the angle is lesser than $180°$, and D_2 represent the sin function of the angle. The Xing Loss is formulated as:

$$L_{Xing} = D_1\big(ReLU(-D_2)\big) + (1 - D_1)\big(ReLU(D_2)\big). \tag{3-8}$$

Note that, while Xing Loss constraints bèzior curves to not self-interact, it does not constraint multiple curves of the same path to interact with each other, thus the regularization ability cannot eliminate the interaction problem completely, Nevertheless, our loss function can be formulated as:

$$L = L_{MSE}(\boldsymbol{I}, \boldsymbol{I}') + \lambda L_{Xing}(\boldsymbol{S}), \tag{3-9}$$

where $\boldsymbol{I}$ is the target image, $\boldsymbol{I}'$ is the rendered image from current SVG $\boldsymbol{S}$, and λ is a constant as a scaling hyper-parameter. We set $\lambda = 0.02$ in our experiments. Furthermore, we also tested a variation of SAMVG using a perception-based metric proposed by Zhang et al. [41]coined Learned Perceptual Image Similarity (LPIPS). LPIPS measures similarity images by l_2 distance on features extracted by a deep learning model. Theoretically, LPIPS loss should be able to capture differences between the target and our render that are more in line with human perception, which cannot be captured using mean square error. In our experiments, we use AlexNet[42] as a feature extractor to get LPIPS Loss. The new loss function is consequently a weighted sum of LPIPS Loss and the previous loss:

$$L' = L_{MSE}(\boldsymbol{I}, \boldsymbol{I}') + \lambda L_{Xing}(\boldsymbol{S}) + \mu L_{LPIPS}(\boldsymbol{I}, \boldsymbol{I}'), \tag{3-10}$$

empirically, we set $\mu = 1.0$ in our experiments.

In SAMVG, we fix the alpha values to one by default, which reduces the complexity of the generated image and further emphasizes the key topologies of the original image. This approach differs from that used by Ma et al. [6], where alpha values were allowed to vary. However, we also include a version of SAMVG with learnable alpha values as a variation in our experiments. The optimization process is performed iteratively for a fixed number of iterations, which we set to 500. We use a point learning rate of 1 and a color learning rate of 0.01 to adjust the SVG parameters during each iteration. In the majority of cases, the SVG produced is already largely in-line with our target after this stage.

3.5 Identifying missing components

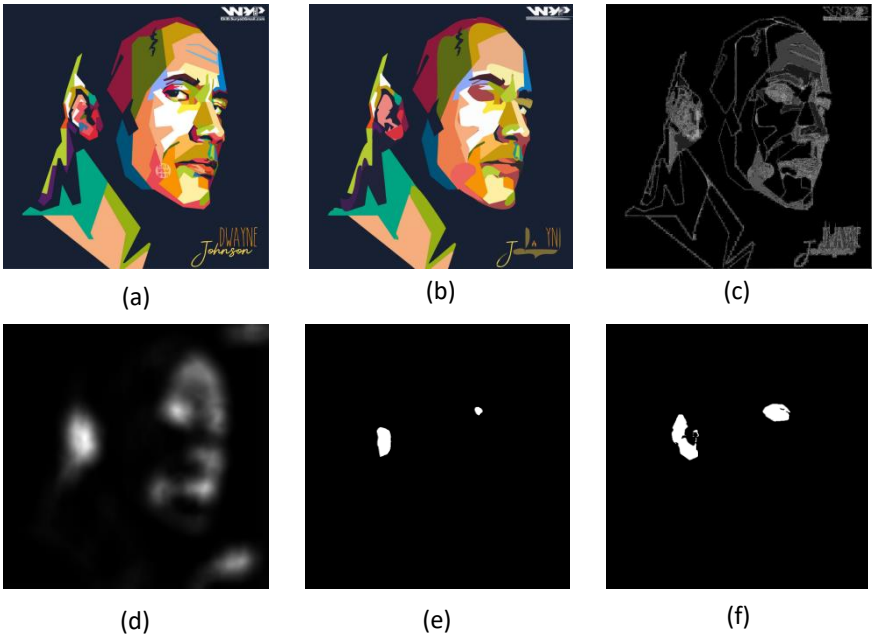

Figure 3-6: An example of identifying the missing components. (a): the target image. (b): the current optimized SVG. (c): the difference map. (d): the difference map after convolution. (e): the difference map after thresholding. (f): the segmented mask using center of components in the threshold map.

After the first phase of optimization, SAMVG may still miss some semantically significant components if their size is too small. These components may carry semantic meanings that are crucial for human perception of the image. For example, SAMVG may occasionally fail to display eyes when vectorizing portrait images due to their small size, as shown in Fig 3-6-a and Fig 3-6-b. However, when prompted at the correct location, SAM can provide appropriate segmentations for these components. We find that there are two possible reasons for this:

1. The initial point grid used as prompts to SAM is not dense enough, and the component does intersect with any of the points in the grid.
2. There are not enough shapes in the initially traced SVG to represent the image in terms of minimizing the mean square error, resulting in the initial shape representing the component being lost in the optimization process to represent a larger component or sub-component.

A naïve way to resolve both of these issues is to significantly increase the density of the point grid, and then filter the generated masks using the "Filter by Impact" method described in Sec. 3.2.1. While this approach may be able to find all components, increasing the grid density is inefficient, as the inference time of SAM is factored by the square of the point grid density. Using LPIPS loss theoretically alleviates the issue caused by the loss of semantically significant components during the optimization process. However, during testing, we observed that relying solely on LPIPS is not robust enough, necessitating a more general solution to ensure that these components are not missed.

Our solution is to detect for these missed components at the end of the first optimization phase by performing convolution on the difference map. Specifically, given the target image $\boldsymbol{I}$ and current render $\boldsymbol{I}'$ from $\boldsymbol{S}$, we compute the difference map $\boldsymbol{D} \in \mathbb{R}^{w \times h}$ summed across color channels. We convolute the difference map with a fixed circular kernel similar to process described in Sec. 3.2.2 to remove noise, arriving at $\boldsymbol{D_1}$. Subsequently, thresholding is applied to $\boldsymbol{D_1}$ such that

$$\boldsymbol{D_2}_{x.y} = \begin{cases} 1, \text{ if } \boldsymbol{D_1}_{x.y} \geq \omega \\ 0, \text{ if } \boldsymbol{D_1}_{x.y} < \omega \end{cases}, \tag{3-11}$$

where $\omega \in \mathbb{R}$ is the threshold value set to be 0.784. Following this, we use the centers of components in $\boldsymbol{D_2}$ as prompts to SAM to retrieve the masks. Finally, the masks are filtered by impact with respect to $\boldsymbol{I}'$ as the starting canvas, then traced and optimized for another 500 iterations to achieve the final SVG.

3.6 Summary

In this chapter, we have provided a detailed description of each procedural step of SAMVG, including segmentation, tracing, and optimization. SAMVG is a multi-stage pipeline that cycles through the "segment-trace-optimize" process twice, with the second cycle correcting any mistakes from the end of the first cycle.

We use a custom filtering method to ensure that only significant shapes are kept, minimizing image complexity, and convolution to locate missing regions from the

segmentations. Compared to adding and optimizing paths layer-by-layer, our algorithm only has two optimization phases, making it more efficient. In general, a 512×512 image takes roughly one to two minutes to complete, depending on the hyperparameters set and the complexity of the image.

Overall, the multi-stage pipeline of SAMVG, combined with its custom filtering and convolution-based approaches, allows for the generation of high-quality and representative vector graphics that accurately capture the topological features of the original image. In the next chapter, we present the experimental results of our framework and compare it to other state-of-the-art methods.

Chapter Four Experiments and Results

In this chapter, we evaluate the performance of SAMVG against other optimization-based vectorization algorithms and experiment with different variations of SAMVG to discover their effects. We evaluate the results both quantitatively and qualitatively to gain a more comprehensive understanding of the reasons behind the differences or similarities in performance. To quantitatively evaluate the performance of SAMVG, we use several metrics, including the Mean Square Error (MSE), the Learned Perceptual Image Patch Similarity (LPIPS), and the Fréchet Inception Distance (FID). We also compare the complexities of the images via the number of SVG parameters and compression ratio. Moreover, we compare the execution time to assess the efficiency of the algorithm. To qualitatively evaluate the performance, we visually compare the generated vector graphics to the original images and assess how well the vector graphics capture the topological features of the original images. By combining both quantitative and qualitative evaluations, we aim to provide a more comprehensive understanding of the strengths and weaknesses of SAMVG and its variations compared to other state-of-the-art vectorization algorithms.

4.1 Experiment prerequisites

4.1.1 Baseline methods

We compare SAMVG to two other optimization-based vectorization algorithms: DIFFVG[8], which introduced differentiable rendering, and LIVE[7], which is currently considered the state-of-the-art algorithm for optimization-based image vectorization. which stands for Layer-wise Image Vectorization was introduced as the current state-of-the-art for optimization-based image vectorization algorithm.

The largest difference between our method, DIFFVG, and LIVE lies in the initialization process. In DIFFVG, all paths are randomly initialized at once and optimized simultaneously with mean square error. In LIVE, circular paths are initialized based on the largest difference

between the current render and the target image, and paths are added iteratively, with each set of paths optimized to convergence before the next set is added on top of the previous paths. This is why the authors call it "layer-wise vectorization". Additionally, LIVE uses a signed distance function to weight the mean square error such that errors closer to the edge of the paths are more important, and it adds a shape-regulating loss called Xing loss to punish self-interaction of the shapes.

In contrast, SAMVG adds paths in two passes, with the paths initialized based on the segmentation masks provided by SAM. We also utilize the Xing loss proposed in LIVE, but use the standard mean square error loss instead of the weighted loss, as we found that the distance-weighted loss decreases optimization speed without bringing significant improvements to the final results.

4.1.2 SAMVG variations

In addition to comparing SAMVG to DIFFVG and LIVE, we also experiment with different variations of SAMVG to evaluate their effects on the performance of the algorithm. For the default setting of SAMVG, we set every path to have a fixed number of segments to be comparable to the baselines in terms of the number of SVG parameters, and we only use MSE Loss and Xing Loss in the optimization process. For the first variation of SAMVG, we add LPIPS Loss on top of the two losses to hopefully generate graphics that are more in line with the target in terms of human perception. For the second variation, we enable learning alpha values for the fill color of the shapes for greater expressivity of the image. For the third and final variation, we use a variable number of segments for each mask component depending on a threshold with the curvature scores in the edge mask (see Sec. 3.3).

4.1.3 Experiment setup

We conduct all experiments using the Adam optimizer for both SAMVG and the baseline algorithms. The experiments are performed on the same platform, with a single Nvidia GeForce RTX 3090 GPU. We set the learning rate for color parameters and point parameters to be 0.01 and 1, respectively, for all algorithms. The number of iterations for

each phase in SAMVG is 500, for a total of 1000 iterations. To ensure a fair comparison, we set the number of iterations in DIFFVG to 1000. For LIVE, we add a single path for each loop and optimize for 100 iterations for each addition, and the total number of iterations depends on the number of total paths. We set the total number of paths in DIFFVG and LIVE based on the number of paths generated by SAMVG for the specific target image. In most cases, the total number of iterations in LIVE is well above 1000, as most images consist of more than 10 paths. By using these consistent experimental settings across all algorithms, we aim to provide a fair and unbiased comparison of the performance of SAMVG and its variations against the baseline algorithms.

4.1.4 Datasets

We evaluate the methods on two different datasets for different comparisons. The first dataset is self-collected from open-source image websites[43, 44][43, 44][61, 62][50, 51][48, 49][48, 49][48, 49][48, 49]. We collected 120 images, with 20 images from 6 categories: Artworks consisting of paintings with different mediums, including oil paintings and watercolors; Emojis consisting of emote icons used in text messaging; 3D renders of various objects using 3D graphic engines; Photographic images of sceneries; Photographic images with a main subject as the focus; and lastly, a style of portrait art known as WPAP (Wedha's Pop Art Portrait) consisting of sharp lines and vibrant color blocks.

Based on the number of paths generated by the default SAMVG, we filtered out images that have more than 128 paths to save experiment time, as LIVE's vectorization time increases exponentially with increasing number of paths. The filtered dataset has a hundred images remaining, and the distribution of images in each category is as follows: photographic scenery (14 images), photographic subject (19 images), artworks (13 images), 3D renders (19 images), emoji (20 images) and portrait pop arts (15 images). The purpose of this dataset is to study the performance of SAMVG under a diverse set of domains, ranging from abstract art to complex scenery. We down-sample high-resolution images to further reduce experiment time. In this dataset, we evaluate the performances by quality and simplicity of the final result and vectorization time. See Sec. 4.2 for more details.

To further study the representation quality of the vectorization, we also evaluate the

performance of classification models on the SVGs generated by different methods. Specifically, we randomly sample 10 images from each class in Imagenette[45] to create a small subset of the classification dataset with 100 images in total. Another challenge of this dataset is the small image sizes, which are known to cause difficulties for vectorization algorithms. We train a standard ResNet classification model on the original dataset and evaluate the classification accuracy of the model on the rasterized version of the SVGs generated.

By using these two datasets, we aim to evaluate the performance of the different vectorization algorithms on a diverse set of images and to assess the quality and accuracy of the generated vector graphics compared to the original images.

4.2 Evaluation metrics

We use various different evaluation metrics to comprehensively analyze the vectorization results in different aspects. In this section, we introduce each metric used and the motivation behind using these metrics.

4.2.1 Mean Square Error

Means square error (MSE) measures the average squared difference between the predicted and target values in an image, it is the most straight forward yet important metric for comparing image similarity, and is also the major component for the loss function in the optimization process.

4.2.1 LPIPS

As mentioned in[41], mean square error (MSE) does not behave exactly like human perception when comparing images. For example, if we have a pair of identical images but increase the brightness of one, MSE will report a large error even though humans perceive the two images as similar. The Learned Perceptual Image Similarity (LPIPS) is behaviorally similar to human perception in this regard, as it compares similarity in the feature space

extracted by deep convolution networks. LPIPS has been shown to be more aligned with human perception than traditional pixel-based metrics such as MSE. Note that we report LPIPS Loss, thus lower means more similar. We use the same AlexNet to retrieve loss values, which is identical to the loss function used in our LPIPS variation of SAMVG.

4.2.2 Fréchet Inception Distance

The Fréchet Inception Distance (FID) is another popular method for evaluating generative models[46]. Given two probability distributions $\zeta, \eta \in \mathbb{R}^n$ with finite mean and variances, their Fréchet distance is defined by

$$d_F(\zeta, \eta) \coloneqq \sqrt{\inf_{\gamma \in \Gamma(\zeta,\eta)} \int_{\mathbb{R}^n \times \mathbb{R}^n} ||x - y||^2 \mathrm{d}\gamma(x, y)}, \tag{4-1}$$

where $\Gamma(\zeta, \eta)$ is the set of all measures on $\mathbb{R}^n \times \mathbb{R}^n$ with marginals ζ, η on the first and second factors respectively. In the context of two dataset of images, we transform the images into feature vectors of dimension n (2048 by default) using the Inception V3 model[47] trained on the ImageNet[48], we then fit two gaussian distributions $N(\zeta, \Sigma)$ and $N(\zeta', \Sigma')$ for the two datasets respectively. The Fréchet distance can be explicitly solved as[49]

$$d_F(N(\zeta, \Sigma), N(\zeta', \Sigma'))^2 = ||\zeta - \zeta'||_2^2 + \mathrm{tr}\left(\Sigma + \Sigma' - 2\sqrt{\Sigma^{\frac{1}{2}} \cdot \Sigma' \cdot \Sigma^{\frac{1}{2}}}\right). \tag{4-2}$$

Compared to LPIPS, FID measures the similarity of the entire generated dataset. By using FID as an evaluation metric, we aim to assess the overall similarity between the generated dataset and the original dataset, and to provide a more comprehensive analysis of the performance of the different vectorization algorithms.

4.2.3 Classification accuracy

As previously stated, we prepared a small classification dataset to be vectorized for classification and evaluate the drop-off in the performance of a classification model.

Theoretically, this metric is similar to LPIPS Loss, as we are measuring the difference between the SVG and the target using deep models, but at different stages of the encoding process. However, we greatly increase the vectorization difficulty in the classification dataset by using lower image resolutions, which greatly impacts the optimization quality as the image has a more discrete distribution. For the classification model, we train a standard ResNet-101 model[50] with a single fully connected layer for prediction.

4.2.4 Vectorization speed

One of the major disadvantages of the previous state-of-the-art LIVE is the extremely long vectorization time that scales with the number of paths. To compare the speed increase, we measure the average time taken to vectorize each image for the entire dataset. We use clock time difference instead of GFLOPS or other benchmark methods as the vectorization time is long enough (> 60 seconds) such that the inaccuracies are not significant. By measuring the vectorization time, we aim to evaluate the efficiency of the different vectorization algorithms and to provide a more practical assessment of their performance in real-world scenarios where time is a critical factor.

This evaluation metric is particularly important for practical applications such as image editing and graphics design, where fast and efficient vectorization is essential for improving productivity and workflow. Vectorization speed is also important for batch vectorization to create SVG datasets.

4.2.5 Image complexity

Given the same representative ability of an SVG, the simpler graphic is preferred for editability. One simple way of measuring image complexity for SVG is to simply count the number of parameters. As SVGs generated by SAMVG, LIVE and DiffVG only consist of closed bèzior paths with fill colors, we only consider the bèzior control points and fill colors as variable parameters. Even though we set the number of paths and segments of DiffVG and LIVE to follow the default version of SAMVG, the default SAMVG has fixed alpha values for the fixed color, thus the number of parameters for each fill color in SAMVG is 3

instead of 4. Other variations of SAMVG also may have different number of paths as the second cycle may produce more or less new paths based on the previous optimization results.

As another measure of image complexity in raster form, we use a compression-based method to measure complexity[51]. Given an image $\boldsymbol{I}$, we first generate a compressed version of the image $\boldsymbol{I}'$ with the JPEG standard[52]. We calculate the root mean square error RMSE between $\boldsymbol{I}$ and $\boldsymbol{I}'$, and the compression ratio r, the complexity of is defined as

$$complexity(\boldsymbol{I}) = \frac{RMSE}{r} . \tag{4-3}$$

Intuitively, we are measuring how well the image can be compressed with Discrete Fourier Transform, and images with simpler distribution should be better compressed.

4.3 Quantitative Results

Table 4-1 Quantitative results on the self-collected dataset

Metrics	LIVE	DIFFVG	SAMVG	SAMVG +LPIPS	SAMVG +alpha	SAMVG +var
MSE↓	5.75×10^{-3}	1.61×10^{-2}	5.30×10^{-3}	5.33×10^{-3}	**4.89×10^{-3}**	5.31×10^{-3}
LPIPS↓	2.59×10^{-1}	3.18×10^{-1}	2.49×10^{-1}	2.50×10^{-1}	**2.43×10^{-1}**	2.46×10^{-1}
FID↓	188.54	209.25	186.75	186.00	**184.24**	186.83
Complexity↓	11.53	12.17	11.51	11.50	**11.32**	11.38
Paths↓	59.96	59.96	59.96	**56.31**	57.54	57.68
Number of SVG Parameters↓	2038	2038	1978	**1858**	1956	2343
Time (s) ↓	2609.00	**139.57**	156.00	147.00	142.00	167.61

The quantitative results of the diverse self-collected datasets are summarized and reported in Tab. 1. Note that SAMVG+LPIPS, SAMVG+alpha and SAMVG+var represent the variations of SAMVG with LPIPS loss, learnable alpha values and variable segment number, respectively. All metrics are reported as average values across the dataset, except for FID, which is measured by directly comparing the generated dataset and the original

dataset. Next, we report these metrics on each category in the dataset.

Table 4-2 Quantitative results on Art (13 images)

Metrics	LIVE	DIFFVG	SAMVG	SAMVG +LPIPS	SAMVG +alpha	SAMVG +var
MSE↓	7.04×10^{-3}	7.71×10^{-3}	6.98×10^{-3}	6.94×10^{-3}	**6.52×10^{-3}**	6.72×10^{-3}
LPIPS↓	4.33×10^{-1}	4.61×10^{-1}	4.33×10^{-1}	4.35×10^{-1}	4.27×10^{-1}	**4.23×10^{-1}**
FID↓	340.12	347.88	338.92	334.02	369.29	**328.39**
Complexity↓	14.70	15.52	14.79	14.75	**14.51**	14.64
Paths↓	68.00	68.00	68.00	**67.54**	67.69	67.85
Number of SVG Parameters↓	2312	2312	2244	**2228**	2301	2919
Time (s) ↓	3011.41	**160.10**	185.35	169.40	168.49	206.24

Table 4-3 Quantitative results on Emoji (20 images)

Metrics	LIVE	DIFFVG	SAMVG	SAMVG +LPIPS	SAMVG +alpha	SAMVG +var
MSE↓	3.87×10^{-3}	1.05×10^{-2}	3.10×10^{-3}	3.32×10^{-3}	**2.79×10^{-3}**	3.46×10^{-3}
LPIPS↓	6.85×10^{-2}	1.45×10^{-1}	7.20×10^{-2}	7.50×10^{-2}	**6.64×10^{-2}**	7.94×10^{-2}
FID↓	145.84	183.74	151.17	145.18	**137.68**	155.04
Complexity↓	5.54	5.72	**5.48**	5.57	5.55	5.51
Paths↓	13.85	13.85	13.85	13.70	**13.60**	13.90
Number of SVG Parameters↓	470	470	457	**452**	462	511
Time (s) ↓	114.88	**32.29**	38.47	35.53	34.45	40.06

Table 4-4 Quantitative results on Render (19 images)

Metrics	LIVE	DIFFVG	SAMVG	SAMVG +LPIPS	SAMVG +alpha	SAMVG +var
MSE↓	4.95×10^{-3}	4.20×10^{-2}	4.83×10^{-3}	4.78×10^{-3}	**4.49×10^{-3}**	4.85×10^{-3}
LPIPS↓	2.84×10^{-1}	4.06×10^{-1}	2.70×10^{-1}	2.70×10^{-1}	**2.67×10^{-1}**	2.68×10^{-1}
FID↓	12.57	13.57	12.61	12.68	12.39	**12.33**
Complexity↓	14.70	15.52	14.79	14.75	**14.51**	14.64
Paths↓	61.53	61.53	61.53	**61.21**	61.42	61.16
Number of SVG Parameters↓	2091	2091	2030	**2019**	2088	2326
Time (s) ↓	2524.66	**152.03**	171.97	164.98	159.12	173.99

Table 4-5 Quantitative results on Scenery (14 images)

Metrics	LIVE	DIFFVG	SAMVG	SAMVG +LPIPS	SAMVG +alpha	SAMVG +var
MSE↓	5.17×10^{-3}	1.50×10^{-2}	5.53×10^{-3}	5.43×10^{-3}	**5.00×10^{-3}**	5.68×10^{-3}
LPIPS↓	3.43×10^{-1}	3.88×10^{-1}	3.35×10^{-1}	3.33×10^{-1}	3.31×10^{-1}	**3.29×10^{-1}**
FID↓	14.41	15.71	14.79	14.66	**14.39**	14.59
Complexity↓	14.70	15.52	14.79	14.75	**14.51**	14.64
Paths↓	63.86	63.86	63.86	64.14	**63.79**	64.07
Number of SVG Parameters↓	2171	2171	**2107**	2116	2168	2539
Time (s) ↓	2804.47	**158.91**	173.89	171.49	157.91	185.77

Table 4-6 Quantitative results on Subjects (19 images)

Metrics	LIVE	DIFFVG	SAMVG	SAMVG +LPIPS	SAMVG +alpha	SAMVG +var
MSE↓	6.17×10^{-3}	7.43×10^{-3}	6.37×10^{-3}	6.39×10^{-3}	**5.82×10^{-3}**	6.35×10^{-3}
LPIPS↓	3.33×10^{-1}	3.67×10^{-1}	3.30×10^{-1}	3.30×10^{-1}	**3.20×10^{-1}**	3.25×10^{-1}
FID↓	13.84	14.20	13.81	13.72	**13.47**	13.64
Complexity↓	14.70	15.52	14.79	14.75	**14.51**	14.64
Paths↓	55.53	55.53	55.53	55.79	**55.37**	55.95
Number of SVG Parameters↓	1887	1887	**1832**	1841	1882	2395
Time (s) ↓	2509.62	**134.10**	159.65	147.26	146.52	177.15

Table 4-7 Quantitative results on Pop Art (15 images)

Metrics	LIVE	DIFFVG	SAMVG	SAMVG +LPIPS	SAMVG +alpha	SAMVG +var
MSE↓	8.18×10^{-3}	9.76×10^{-3}	5.81×10^{-3}	5.92×10^{-3}	5.51×10^{-3}	**5.49×10^{-3}**
LPIPS↓	1.58×10^{-1}	1.84×10^{-1}	1.18×10^{-1}	1.16×10^{-1}	1.12×10^{-1}	**1.09×10^{-1}**
FID↓	9.82	10.22	9.32	9.36	9.32	**9.29**
Complexity↓	14.70	15.52	14.79	14.75	**14.51**	14.64
Paths↓	**97.00**	**97.00**	**97.00**	**97.00**	99.33	99.07
Number of SVG Parameters↓	3298.00	3298.00	**3201.00**	**3201.00**	3377.33	4058.80
Time (s) ↓	5640.47	**199.96**	245.42	228.74	220.11	267.06

The classification accuracy and loss values on the Imagenette subset are reported in Tab. 8. “Target” refers to the model’s performance on the target images. The loss values are cross-entropy loss on the final prediction probabilities of the model. It is shown that SAMVG has the edge in terms of both classification accuracy and loss over the baselines, except for the SAMVG+var variation, which will be discussed in later sections.

Table 4-8 Classification accuracy and loss

Metric	**Target**	LIVE	DIFFVG	SAMVG	SAMVG +LPIPS	SAMVG +alpha	SAMVG +var
Accuracy↑	**0.90**	0.61	0.41	**0.64**	0.62	**0.64**	0.54
Loss↓	**0.011**	0.028	0.042	0.028	**0.027**	0.028	0.033

4.3.1 Comparison against LIVE and DIFFVG

In this section, we focus on comparing the default version of SAMVG against the baselines LIVE and DIFFVG, as they are equal in terms of the number of path segments. SAMVG outperforms both LIVE and DIFFVG across all measures of vectorization quality, including MSE, LPIPS, and FID. Additionally, our method achieves the highest classification accuracy compared to the baselines, demonstrating high representational ability even for low-resolution images. Unsurprisingly, our method is also the least complex as it does not include opacity features. In terms of vectorization time, SAMVG is an order of magnitude faster than LIVE. Our method is slightly slower than DIFFVG because of the additional inference time needed to generate the masks. Nevertheless, SAMVG achieves significantly better vectorization quality for a small sacrifice in speed. Note that the speed increase scales even more as the images get larger and more complex, as shown in Fig. 4-1, SAMVG's algorithm has a lower time complexity compared to LIVE, which makes it more suitable for applications where fast and efficient vectorization is essential.

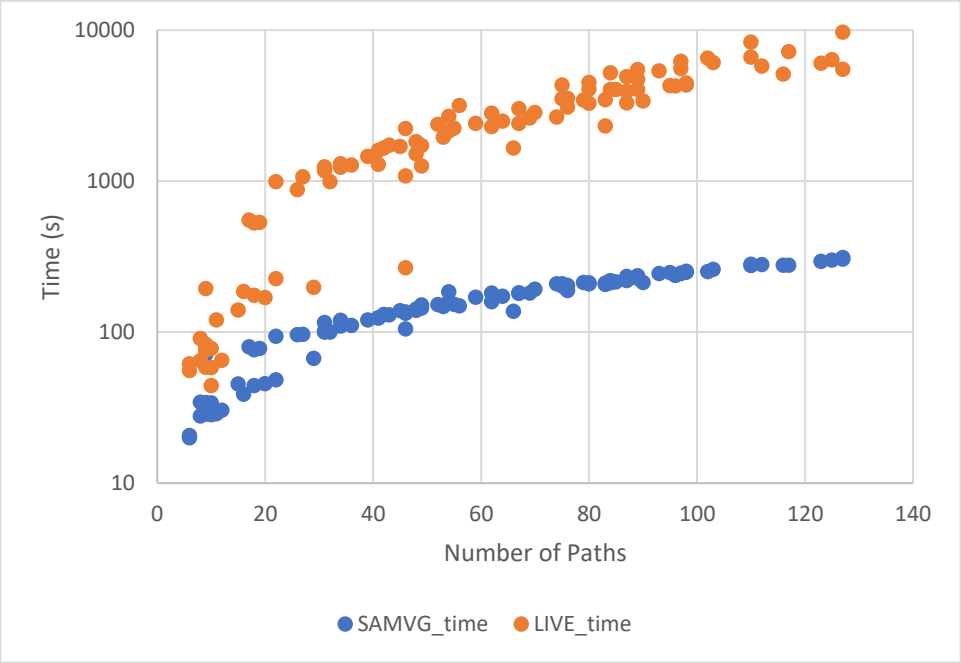

Figure 4-1: Number of paths vs. time taken by each algorithm. The Y-axis (time in seconds) is shown in logarithmic scale.

SAMVG has particularly larger improvements over LIVE in the Emoji and Pop Art categories, which are the most "Clip Art" like categories. We hypothesize that the reason for this is that each component of images in these categories is often of a single color, making the initial traced shapes from the masks even closer to the target image.

4.3.2 Effect of adding LPIPS loss

Unexpectedly, adding LPIPS loss did not significantly improve the LPIPS loss. The difference in other metrics is too small to be considered statistically significant. We suspect that the current LPIPS loss does not translate well in terms of gradient descent of the SVG parameters, although further testing is required for conclusive statements. However, the variation of SAMVG with LPIPS loss does have the least number of paths on average. Because this variation behaves exactly the same as the default up to the optimization stage, having fewer paths suggests that this variation converged faster, such that fewer paths are added in the second stage.

4.3.3 Effect of learnable alpha values

This variation performed the best overall, as having variable opacities greatly increased the expressivity of the vector graphic. Moreover, having translucent fill colors allowed for larger margins of error in the segment and trace steps of SAMVG, because shapes on the lower layers cannot be properly optimized if they are invisible due to being covered by shapes on top of it. Surprisingly, this variation also produced graphics with the least complexity in terms of the number of parameters and compression ratio. Most likely, this variation also converged much better at the end of the first optimization phase and required fewer additional paths. However, it is important to note that conventional vector graphics that humans produce usually do not consist of translucent shapes. Though subjective, we show later in qualitative analysis some undesirable effects of variable alpha values. This highlights the importance of considering the trade-offs between expressivity and simplicity in vectorization algorithms, and the need for further research to develop more effective strategies for balancing these factors.

4.3.4 Effect of variable segment numbers

Using a variable number of segment numbers slightly increased the number of parameters for the SVGs. In terms of vectorization quality, this variation had the best performance in the Art and Pop Art categories, which also happen to be categories with the greatest number of paths. We suspect that the varying segment paths play a greater role in images with high structural complexities that have more complex shape primitives. However, no conclusive statements can be drawn as both the difference in performance and the sample size are small.

We also observe that this variation has a large drop-off in performance in the classification dataset. As mentioned before, one of the challenges in this dataset is the low target image resolution. Therefore, this result suggests that the threshold-based corner selection algorithm does not adapt well for small-sized images.

4.3.4 Visualization of performance under different categories

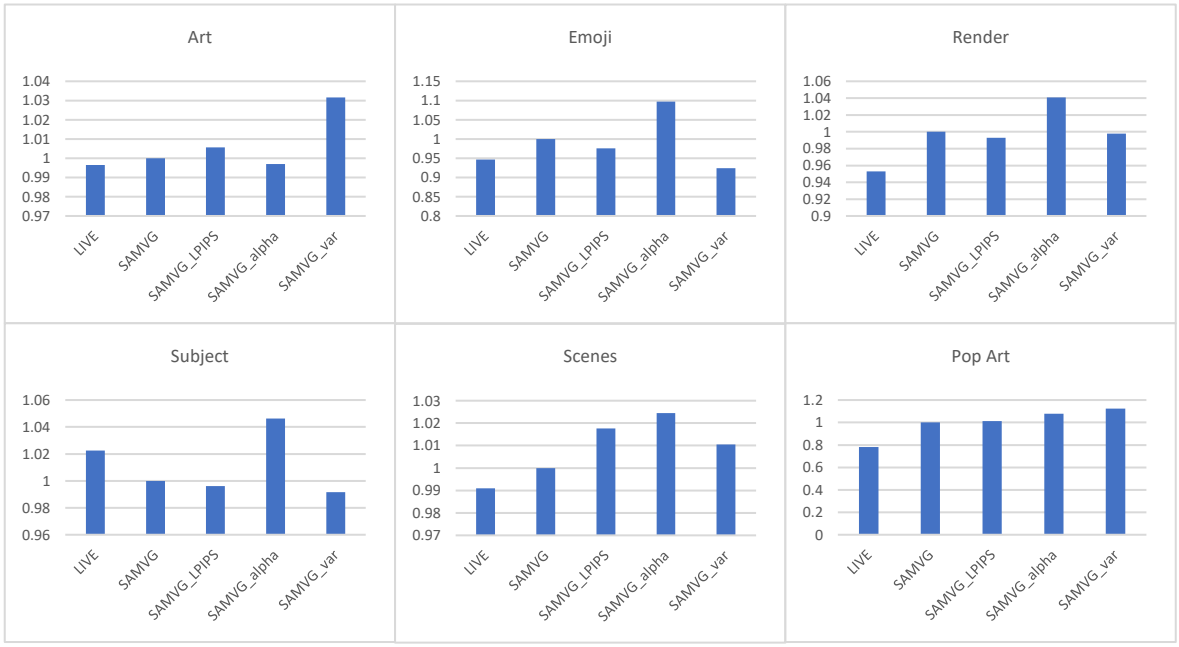

Figure 4-2: Bar charts comparing the vectorization scores between different methods under each category.

To better visualize the difference in vector quality of each method, we compute a vectorization score based on the normalized sum of MSE, LPIPS, and FID. Specifically, we normalize the MSE, LPIPS, and FID with the default SAMVG as the baseline for each method tested in the experiment, except for DIFFVG, as its metric scores are significantly worse compared to the rest of the methods. We sum these normalized scores and take their inverse as the normalized score indicating the vectorization quality of each method. Higher scores indicate better vectorization quality. These scores are computed separately for each category in the dataset to expose the effects of different domains and are visualized in the form of bar charts, as illustrated in Fig. 4-2.

The bar charts provide a clear and concise overview of the performance of each vectorization method across different categories of images. We find that the default SAMVG outperformed LIVE across all categories except for the subject category, indicating the effectiveness of the proposed method. These findings are consistent with our earlier

observations that adding variable opacity to the SVGs greatly increases the expressivity of the vector graphics, leading to higher-quality vectorizations. However, it is important to note that conventional vector graphics that humans produce usually do not consist of translucent shapes. We explore the effect of the translucent effect further in the next section of qualitative analysis, and discuss the possible improvements that can be made for SAMVG in the final chapter.

The visualization of the vectorization scores confirms our previous observations that the variable number of segments variation of SAMVG tends to perform better with images with more complex shapes. This is evident from the extreme difference in scores between the Emoji and Pop art datasets, where the Pop art dataset achieved a much higher score than the Emoji dataset. In terms of image style, Emoji and Pop art are essentially the same, with discrete color components compositing the image. The main difference between the two is the number and complexity of the components, where Pop arts are complex while Emojis are the simplest among all the categories. This finding suggests that the varying segment paths play a more significant role in images with high structural complexity and more complex shape primitives.

The results of our analysis reveal that the learnable alpha value variation has the best performance overall, achieving the highest scores in four out of the six categories. However, it is weaker than SAMVG+var in the Pop art category and weaker than all other variations of SAMVG in the Art category, suggesting that it may not be well-suited for complex abstract images.

Overall, our analysis shows that SAMVG has better vectorization quality compared to LIVE, with better scores in five out of the six categories. Furthermore, SAMVG+alpha is the best variation in general, achieving the highest scores in four out of the six categories and performing consistently better than LIVE across all categories. These findings highlight the effectiveness of the proposed variations of SAMVG in improving the quality of vectorizations and provide insights into the factors that affect the performance of vectorization algorithms. By understanding these factors, we can develop more effective strategies for vectorizing different types of images and graphics and improve the overall quality of vectorizations.

4.4 Qualitative analysis

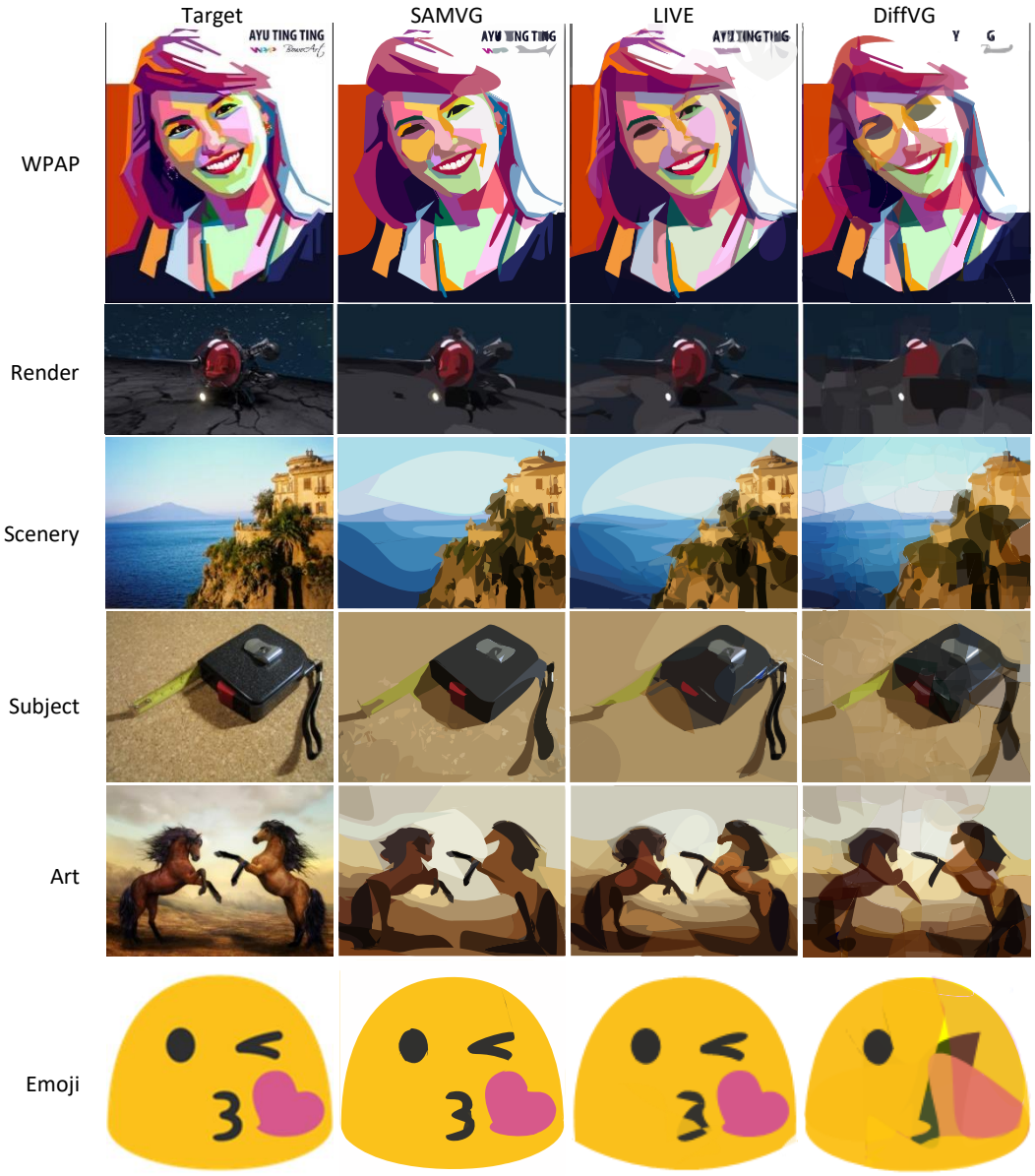

Figure 4-3: Qualitative comparisons between the default SAMVG vs, DIFFVG and LIVE. A sample is taken from each category in the diverse dataset. The reader can zoom in for details in the electronic version of this paper. Best viewed in color.

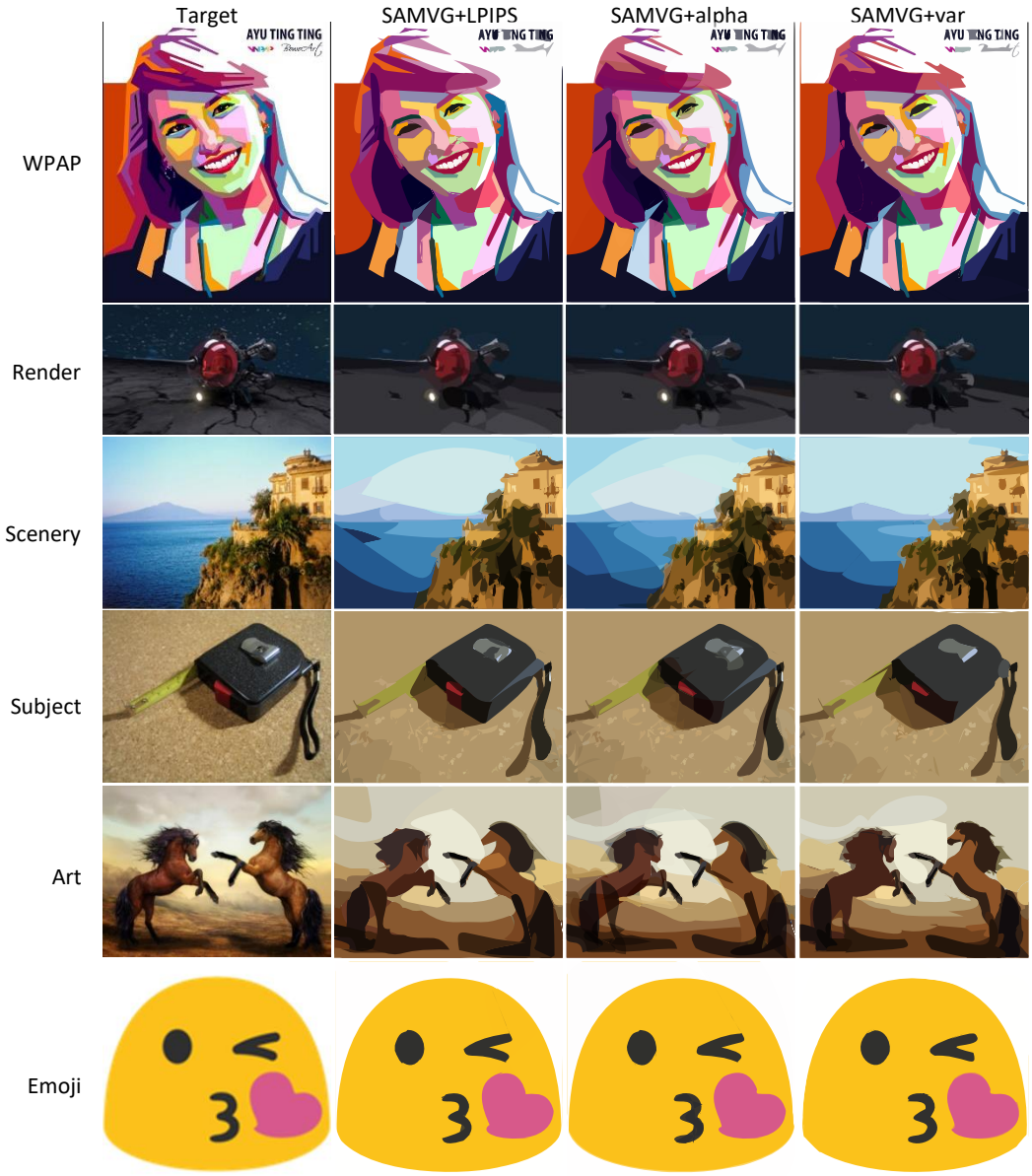

Figure 4-4 Qualitative comparisons between the variations of SAMVG. A sample is taken from each category. The reader can zoom in for details in the electronic version of this paper. Best viewed in color.

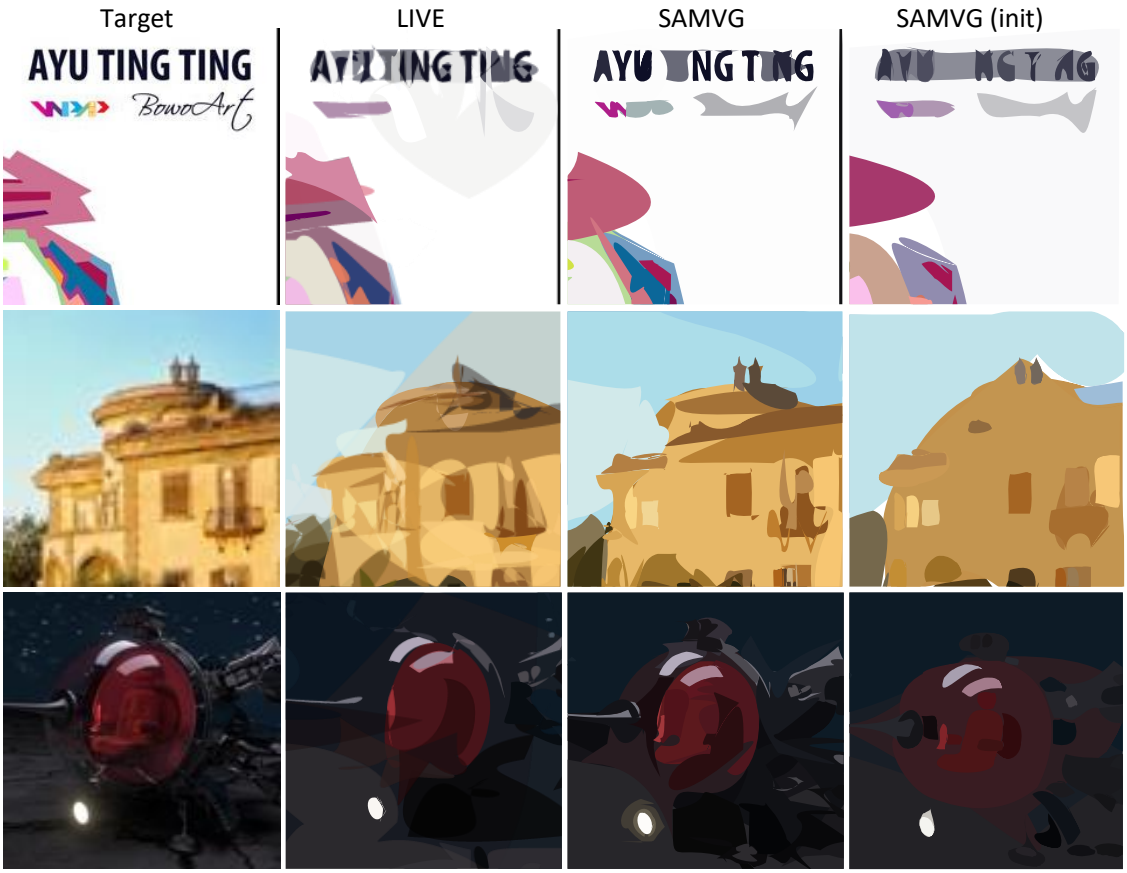

Figure 4-5: Zoomed-in Comparisons of LIVE, SAMVG and the initial trace (before optimization) of SAMVG. Best viewed in color. Graphics are from the self-collected dataset.

From the side-by-side comparison in Fig. 4-3, we see that SAMVG generated vector graphics that represent the target image well. Compared to LIVE, the graphics generated by LIVE is slightly cleaner with lesser meaningless shapes (see Fig. 4-5 for zoomed-in comparisons). On the other hand, DiffVG produced patched-liked colors to compose the image, as their initial paths are randomly generated and thus uniformly distributed throughout the canvas.

However, we observe that SAMVG occasionally performs worse than LIVE if the segmentation model fails to provide good segmentations. As we can see in the "Art" category in Fig. 4-3, SAMVG failed to provide shading details of the horse and gave an arguably worse result than DiffVG. In fact, SAMVG had worse MSE and LPIPS scores than the other baselines for this image in particular. Upon further analysis, we identified two main reasons

for the worse result. Firstly, SAMVG did not generate masks for the shading of the horses in the first stage, as SAM identifies the horse to be one segmentation. Instead, SAM segmented the legs and the hoofs of the horses, which were actually more semantically correct segmentations compared to the shading. However, the shading is more visually impactful than differentiating the legs and the hoofs in this case. Secondly, as the shading is similar in color, it was not detected during the missing component search phase (Chapter 3.5). We find that results like this to be rare, but more comprehensive testing with larger datasets is needed to study the robustness of SAMVG.

The difference between LIVE and SAMVG in terms of vectorization quality is subtle and subjective to the viewer's preference. Nevertheless, it is fair to say that SAMVG is at least on par with LIVE in terms of vectorization quality and far superior in terms of speed. Furthermore, SAMVG is better at retrieving semantic features from the target image. As shown in Fig. 4-5, the initially traced SVG from the masks picks up objects that are semantically important but difficult to detect using color-based operations, such as objects that are either small in size (e.g., Fig. 4-5, second row, the chimneys on the rooftop) or has a similar color to their background (e.g., Fig. 4-5, third row, the seats in the spacecraft).

This ability to retrieve semantic features from the target image is a key advantage of SAMVG over other vectorization methods, as it allows for more accurate and meaningful representations of the image.

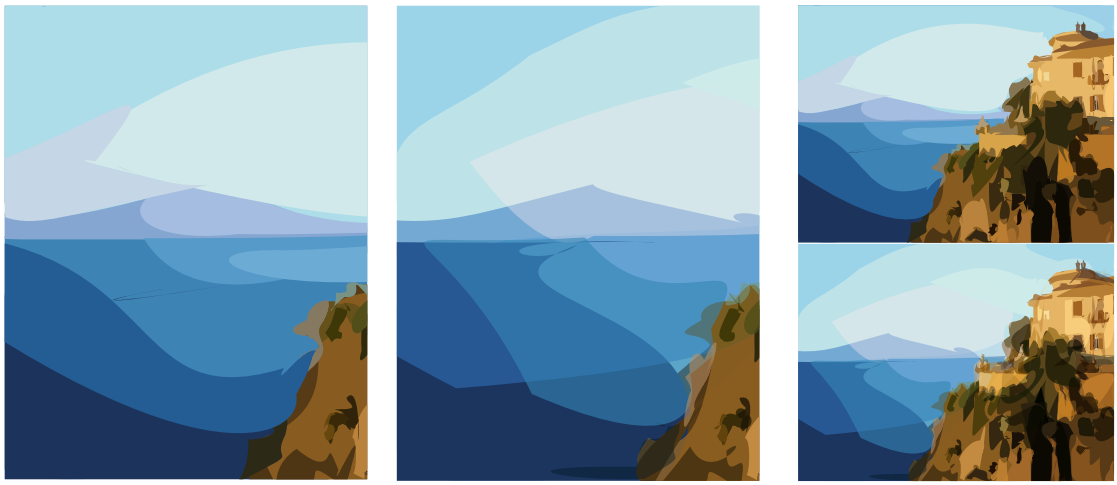

Figure 4-6: Close-up comparison of SAMVG (left) vs. SAMVG+alpha (mid-right) and the original graphic of SAMVG (top right) and SAMVG+alpha (bottom right). Graphics are from the self-collected dataset.

Our analysis of the variations of SAMVG reveals several interesting findings. Firstly, we find no observable difference in the LPIPS variation, suggesting the current version of LPIPS loss may be unimpactful to the optimization process. Secondly, the variable opacity version of SAMVG does produce richer images by laying translucent shapes on top of each other. As shown in Fig. 4-6, using translucent shapes, the SVG is able to represent multiple color components using a lesser number of paths, essentially creating the illusion of the graphic having more shape primitives than it actually has. Although the translucent effect does produce images closer to the target image, especially for photographic images, the SVG generated is more difficult for a human to edit as each shape is more interlaced with each other, and differs from how a human would vectorize an image manually. Thirdly, comparing the variation of SAMVG with a variable number of segments against the default version with a fixed number of segments, we find the variable version to be better at capturing more complex shapes, although it seemed to have more artifacts, possibly due to the less regularized path initialization. We also find that this variation is considerably better at capturing texts present in the target image, which is not surprising given that this variation of SAMVG is closer to the reference algorithm described in the original literature[1], which was designed for character font vectorization.

Based on our analysis of the variations of SAMVG, we believe that using a variable

Figure 4-7: Illustration of how SAMVG+var is better at capturing shapes such as texts from the target image. Image is sample from the self-collected dataset. The second row is the zoomed-in version of the top row.

number of segments has more potential to produce higher-quality vectorizations given further improvements to the initial trace algorithm and the optimization algorithm. Although this variation of SAMVG performed slightly worse than the fixed segment version in some categories, we find that it is better at capturing more complex shapes and texts present in the target image. This suggests that the variable segment approach has more flexibility in representing different types of images and graphics, and may be better suited for applications that require more expressive vectorizations. In the research outlook section in Sec. 5.2, we discuss these issues further and suggest several directions for future research.

4.5 Effect of target image resolution

In this section, we briefly study the effect of different image resolutions on the vectorization quality of SAMVG. Intuitively, images with low resolution lead to worse vectorization quality as the image becomes more discrete, which is bad for tracing and optimizing. To explore the extent of the deterioration of vectorization quality, we compared

the vectorization results of a single target image with different resolutions. Specifically, we vectorized a 450×600 image with SAMVG and compared the vectorization quality and time taken for each down-sampled resolution of 225×300 and 113×150. For this experiment, we use SAMVG with both variable segments and variable alpha values. The quantitative and qualitative results can be seen below in Fig. 4-8 and Tab. 4-9.

Table 4-9 Quantitative Metrics

Metric	450×600	225×300	113×150
MSE↓	0.053	0.059	0.093
Time (s)↓	256	186	147

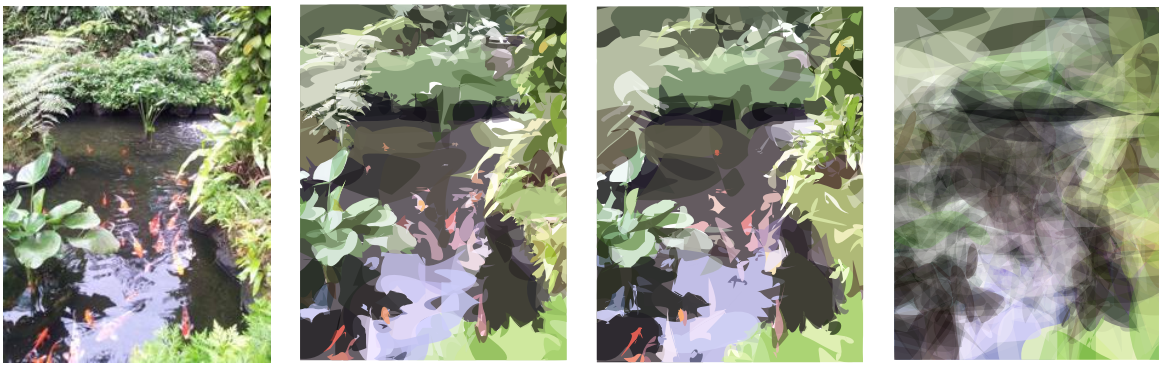

Figure 4-8: Comparisons of SAMVG with different input target resolutions.

The results of our analysis show that the vectorization quality of SAMVG deteriorates significantly as the image resolution continues to decrease to 113×150. This is due to two stages of SAMVG failing as a result of the small image size. Firstly, the tracing algorithm failed as the number of sample points to fit the Bezier curves became too small, causing difficulties in selecting the corner points and tracing appropriate Bezier curves. Secondly, the optimization algorithm failed to find the local maxima for the control points in between the pixels as the learning rate in optimization was set to 1.0.

In contrast, the 225×300 resolution image only has a marginal decrease in vectorization quality, capturing the majority of the important shapes with some loss of details in each shape. This suggests that SAMVG is robust to changes in image resolution up to a

certain point, but may struggle with very low-resolution images.

In terms of vectorization time, there is a correlation between image size and speed, as a large portion of the optimization operation is rendering the vector graphics. Based on this experiment, we recommend that high-resolution images are down-sampled to a reasonable size before batch processing to achieve a balance between fast vectorization speed and good vectorization quality.

These findings provide important insights into the performance of SAMVG across different image resolutions and highlight the need to consider image size and resolution when selecting a vectorization method for a specific application. By understanding these factors, we can develop more effective vectorization strategies that are tailored to the specific needs of different applications and users.

4.6 Summary

In this chapter, we have evaluated the performance of SAMVG over other optimization algorithms in terms of vectorization quality and speed with a multitude of different quantitative metrics. We have also showcased qualitative comparisons between the algorithms and analyzed rare failure cases of SAMVG.

Overall, our results demonstrate that SAMVG is a highly effective vectorization algorithm that is capable of capturing semantic information from the target image in the final vector graphic. We have also studied the variations of SAMVG and found that the alpha value-learning version has the best vectorization quality overall due to its superior expressivity, although we argue that SVG generated from this variation is harder for humans post-editing.

Furthermore, we have analyzed the effect of varying image resolution of the target image and provided an explanation for the results. Our findings suggest that SAMVG is robust to changes in image resolution up to a certain point, but may struggle with very low-resolution images.

Chapter Five Conclusions

5.1 Main conclusions

In this dissertation paper, *Vectorize Anything using Deep Segmentation and Direct Optimization*, we have proposed a novel image vectorization algorithm, SAMVG, that combines the latest deep learning segmentation technology, traditional path tracing vectorization, and direct optimization with differentiable rendering. We began by providing background information on image vectorization, including its history and the demand for a vectorization algorithm that is able to generate SVG graphics with minimal parameters. We also introduced the current state-of-the-art methods and discussed their advantages and limitations. Notably, the previous state-of-the-art, optimization-based algorithm was LIVE, which is capable of producing high-quality, semantically representing SVG given a target image from any domain. However, the main drawback of LIVE was its slow vectorization speed, which motivated us to develop a more efficient algorithm that could achieve comparable or better vectorization quality. To address this challenge, we developed SAMVG, which unifies the latest Segment-Anything Model, traditional tracing algorithms and optimization with differentiable rendering.

We have described the multi-stage pipeline of SAMVG in detail. To summarize the process of SAMVG, we first obtain a set of impactful segmentation masks through the Segment-Anything Model with custom prompts and filtering. These masks provide a rich representation of the target image, capturing both its global structure and local details. Next, the masks are traced with a modified version of a traditional tracing algorithm to generate an initial SVG. This tracing algorithm incorporates several modifications to improve the quality of the vectorization, including adaptive sampling, curve fitting, and corner detection. The SVG is optimized using differentiable rendering with respect to the target raster image. Finally, additional paths are added if significant components are missing before another phase of optimization. This ensures that the final vector graphic is complete and accurate, capturing all of the relevant information from the target image.

In our experiments, we demonstrate that the vectorization quality of SAMVG is superior to LIVE in the majority of cases. Although there exist some cases where LIVE outperforms SAMVG, the difference is subtle, and we believe that these issues can be resolved with further development of SAMVG. For example, a simple and brute-force solution is to add another cycle of adding more paths, and the vectorization time would still be much lower compared to LIVE. More importantly, we show that SAMVG can be scaled to generate SVGs with more paths as we perform a fixed number of iterations regardless of the number of paths. We believe that the scalability of SAMVG can allow for the creation of more SVG datasets through vectorization, enabling future research with deep learning in the SVG domain.

We implemented different variations of SAMVG and observed the effects of the variations. Our quantitative and qualitative analysis of the performance of the variations provided insight into how we can further improve upon SAMVG for future research, which we discuss more in detail in the next and last section of the dissertation.

5.2 Research outlook

In the previous section, we demonstrated that SAMVG is a highly effective approach to image vectorization, achieving superior vectorization quality and efficiency compared to previous state-of-the-art algorithms. However, there are still several limitations of the current version of SAMVG that must be addressed in order to further improve its performance and applicability.

In this section, we outline each limitation of the current version of SAMVG and discuss possible improvements that can be made. Specifically, we focus on the limitations in terms of vectorization quality, scalability, and user-friendliness, and propose several solutions to address these issues.

5.2.1 Initializing paths for shading and highlights

As discussed in Sec. 4.4, SAMVG often misses shading and highlight regions of an

Figure 4: Demonstration on the lack of paths for color variations within objects. Image is taken from the examples in SAM's original repository.

object in the image as SAM considers them to be the same segment. Ideally, additional paths should be added on top of the base segmentation for each significant highlight and shadow. In our experiments, these issues are dampened as SAMVG borrows paths from other segmentations to represent important shading and highlights during the optimization process. However, this kind of behavior deviates from the original intent of our method as we desire an initial SVG that is already close to the final result, and optimization should be for fine-tuning. We demonstrate this effect in Fig. 4-9, the initial trace perfectly outlined the dog's body, but ignored its shading and highlights, to compensate for this, the optimization process transformed shapes representing the dog's ears and the flooring above the dog into its highlights. To address this issue, a possible solution is to insert another phase before optimization to add shading paths on top of each mask region. This would ensure that important shading and highlight regions are captured in the initial SVG, reducing the need for additional paths during the optimization process. However, more development and thorough testing are required to arrive at a desirable method.

5.2.2 More refined tracing algorithm

Figure 5-1: Comparison between the masks generated (left) vs. the initial trace (right).

Although the requirement for accurate tracing is low given that we optimize the graphics later, a better initial trace should lead to even better results and reduce the number of iterations needed in optimization. The current tracing algorithm is still crude and has a large margin of error, partly due to the high constraints on the number of segments to be comparable to previous baselines. Additionally, the algorithm for the variable number of segments is old and was mainly designed for vectorizing texts instead of generic shapes. As shown in Fig. 5-1, SAMVG has the potential to have a much better initial trace based on the masks generated by the Segment-Anything Model. Therefore, another area of improvement for SAMVG is to survey or develop a better tracing algorithm that can accurately and efficiently trace the masks generated by the Segment-Anything Model. Note that the refined tracing algorithm should still use the minimum number of segments possible to trace any given component. This constraint is important as it ensures that the resulting vector graphics can be efficiently optimized and easy for human editing, improving its practicality for downstream applications such as graphic design or datasets for deep learning models.

5.2.3 Remove irregular shape artifacts in optimization

Despite implementing Xing Loss to reduce self-interaction during the optimization process, irregular shape artifacts remained a common symptom across all methods based on

optimization, including SAMVG, LIVE and DIFFVG (see qualitative comparisons in Sec. 4.4). The main cause of these artifacts is that small areas of the bèzior shapes translate into a single pixel or are completely ignored in the rasterization process of the graphics. Therefore, these areas are not penalized during optimization with the mean square error or other loss functions in the raster domain. While using Xing Loss does reduce the number of self-intersecting bèzior curves, it cannot identify other instances of irregularity in the shapes including intersections between bèzior curves of the same path or overly skinny and sharp shapes. Thus, increasing the weight for Xing loss would not help either (this was demonstrated by the ablation studies in LIVE's paper[7]). A more robust shape regularizing loss function remains to be developed for future research regarding image vectorization. This loss function should be designed to identify and penalize irregularities in the shapes that are not captured by existing optimization-based methods.

Overall, addressing these issues of the current SAMVG represent important areas for future research in image vectorization. We believe that SAMVG is the next step towards efficient optimization-based image vectorization, and look forward to continuing to explore these possibilities in future research in this exciting and rapidly involving field.

References

[1] ITOH K, OHNO Y. A Curve Fitting Algorithm for Character Fonts [J]. Electron Publ, 1993, 6: 195-205.

[2] XIA T, LIAO B, YU Y. Patch-based image vectorization with automatic curvilinear feature alignment [Z]. ACM SIGGRAPH Asia 2009 papers. Yokohama, Japan; Association for Computing Machinery. 2009: Article 115.10.1145/1661412.1618461

[3] SUN J, LIANG L, WEN F, et al. Image vectorization using optimized gradient meshes [J]. ACM Trans Graph, 2007, 26(3): 11–es.

[4] QUINT A. Scalable Vector Graphics [J]. IEEE MultiMedia, 2003, 10(3): 99–102.

[5] ORZAN A, BOUSSEAU A, BARLA P, et al. Diffusion curves: a vector representation for smooth-shaded images [J]. Commun ACM, 2013, 56(7): 101–8.

[6] XIE G, SUN X, TONG X, et al. Hierarchical diffusion curves for accurate automatic image vectorization [J]. ACM Trans Graph, 2014, 33(6): Article 230.

[7] MA X, ZHOU Y, XU X, et al. Towards Layer-wise Image Vectorization [J/OL] 2022, arXiv:2206.04655[https://ui.adsabs.harvard.edu/abs/2022arXiv220604655M. 10.48550/arXiv.2206.04655

[8] LI T-M, LUKÁČ M, GHARBI M, et al. Differentiable vector graphics rasterization for editing and learning [J]. ACM Trans Graph, 2020, 39(6): Article 193.

[9] CARLIER A, DANELLJAN M, ALAHI A, et al. DeepSVG: A Hierarchical Generative Network for Vector Graphics Animation [J/OL] 2020, arXiv:2007.11301[https://ui.adsabs.harvard.edu/abs/2020arXiv200711301C. 10.48550/arXiv.2007.11301

[10] EGIAZARIAN V, VOYNOV O, ARTEMOV A, et al. Deep Vectorization of Technical Drawings [Z]. Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII. Glasgow, United Kingdom; Springer-Verlag. 2020: 582–98.10.1007/978-3-030-58601-0_35

[11] LOPES R G, HA D R, ECK D, et al. A Learned Representation for Scalable Vector Graphics [J]. 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019: 7929-38.

[12] SHEN I C, CHEN B-Y. ClipGen: A Deep Generative Model for Clipart Vectorization and Synthesis [J]. IEEE Transactions on Visualization and Computer Graphics, 2022, 28(12): 4211-24.

[13] REDDY P, GHARBI M, LUKAC M, et al. Im2Vec: Synthesizing Vector Graphics without Vector Supervision, F 2021, 2021 [C]. IEEE.

[14] SU H, NIU J, LIU X, et al. Vectorization of Raster Manga by Deep Reinforcement Learning [J]. arXiv pre-print server, 2021.

[15] YANG M, CHAO H, ZHANG C, et al. Effective Clipart Image Vectorization through Direct Optimization of Bezigons [J]. IEEE Transactions on Visualization and Computer Graphics, 2016,

22(2): 1063-75.

[16] DIEBEL J R. Bayesian image vectorization: the probabilistic inversion of vector image rasterization [D]; Stanford University, 2008.

[17] SELINGER P. Potrace: a polygon-based tracing algorithm [J/OL] 2003, http://potrace.sourceforge.net/potrace.

[18] KIRILLOV A, MINTUN E, RAVI N, et al. Segment Anything [J]. arXiv pre-print server, 2023.

[19] SARFRAZ M, KHAN M. An automatic algorithm for approximating boundary of bitmap characters [J]. Future Generation Computer Systems, 2004, 20: 1327-36.

[20] HOSHYARI S, DOMINICI E A, SHEFFER A, et al. Perception-driven semi-structured boundary vectorization [J]. ACM Trans Graph, 2018, 37(4): Article 118.

[21] FAVREAU J-D, LAFARGE F, BOUSSEAU A. Photo2clipart: image abstraction and vectorization using layered linear gradients [J]. ACM Trans Graph, 2017, 36(6): Article 180.

[22] KIM B, WANG O, ÖZTIRELI A C, et al. Semantic Segmentation for Line Drawing Vectorization Using Neural Networks [J]. Computer Graphics Forum, 2018, 37.

[23] SÝKORA D, BURIÁNEK J, ZARA J. Sketching cartoons by example [J]. 2005.

[24] Vector Magic [Z]. 2023

[25] Adobe illustrator 2022: Image trace [Z]. 2023

[26] ZHAO S, DURAND F, ZHENG C. Inverse Diffusion Curves Using Shape Optimization [J]. IEEE Transactions on Visualization and Computer Graphics, 2018, 24(7): 2153-66.

[27] DUFF T. Polygon scan conversion by exact convolution; proceedings of the International Conference On Raster Imaging and Digital Typography, Lausanne, F, 1989 [C].

[28] FABRIS A. Antialiasing of Curves by Discrete Pre-filtering [M]. 1997.

[29] MANSON J, SCHAEFER S. Analytic Rasterization of Curves with Polynomial Filters [J]. Computer Graphics Forum, 2013, 32(2pt4): 499-507.

[30] VINKER Y, PAJOUHESHGAR E, BO J, et al. CLIPasso: Semantically-Aware Object Sketching [M]. 2022.

[31] VINKER Y, ALALUF Y, COHEN-OR D, et al. CLIPascene: Scene Sketching with Different Types and Levels of Abstraction [M]. 2022.

[32] RADFORD A, KIM J W, HALLACY C, et al. Learning Transferable Visual Models From Natural Language Supervision; proceedings of the International Conference on Machine Learning, F, 2021 [C].

[33] AOKI H, AIZAWA K. SVG Vector Font Generation for Chinese Characters with Transformer; proceedings of the 2022 IEEE International Conference on Image Processing (ICIP), F 16-19 Oct. 2022, 2022 [C].

[34] JAIN A, XIE A, ABBEEL P. VectorFusion: Text-to-SVG by Abstracting Pixel-Based Diffusion Models [M]. 2022.

[35] EFIMOVA V, JARSKY I, BIZYAEV I, et al. Conditional Vector Graphics Generation for Music Cover Images [M]. 2022.

[36] EFIMOVA V, CHEBYKIN A, JARSKY I, et al. Neural Style Transfer for Vector Graphics [M]. 2023.

[37] SU W, ZHU X, TAO C, et al. Towards All-in-one Pre-training via Maximizing Multi-modal Mutual

Information [M]. 2022.

[38] BAO H, DONG L, WEI F. BEiT: BERT Pre-Training of Image Transformers [J]. ArXiv, 2021, abs/2106.08254.

[39] YIZONG C. Mean shift, mode seeking, and clustering [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1995, 17(8): 790-9.

[40] BOX M J. Non-linear optimization techniques; authors: M. J. Box, D. Davies [and] W. H. Swann [M]. Edinburgh: Published for Imperial Chemical Industries Ltd by Oliver & Boyd, 1969.

[41] ZHANG R, ISOLA P, EFROS A, et al. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric [J]. 2018.

[42] KRIZHEVSKY A. One weird trick for parallelizing convolutional neural networks [J]. 2014.

[43] Pixabay [Z]. 2023

[44] FreeImages [Z]. 2023

[45] HOWARD J. Imagenette [Z]. 2023

[46] HEUSEL M, RAMSAUER H, UNTERTHINER T, et al. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium [M]. 2017.

[47] SZEGEDY C, LIU W, JIA Y, et al. Going Deeper with Convolutions [J]. 2014.

[48] DENG J, DONG W, SOCHER R, et al. ImageNet: a Large-Scale Hierarchical Image Database [M]. 2009.

[49] DOWSON D C, LANDAU B V. The Fréchet distance between multivariate normal distributions [J]. Journal of Multivariate Analysis, 1982, 12(3): 450-5.

[50] HE K, ZHANG X, REN S, et al. Deep Residual Learning for Image Recognition [M]. 2016.

[51] MACHADO P, CARDOSO A. Computing Aethetics [M]. 1998.

[52] WALLACE G K. The JPEG still picture compression standard [J]. Commun ACM, 1991, 34(4).

Acknowledgements

I would like to express our sincere gratitude to Ran Yi, our dissertation supervisor, for providing invaluable guidance and support throughout this research project. We also extend our thanks to Paul Rosin and Yukun Lai for their insightful discussions and helpful feedback, which greatly contributed to the success of this work. Additionally, we would like to acknowledge the valuable feedback provided by Hu Teng on the composition of this dissertation. Their contributions have been instrumental in shaping the final outcome of this study.