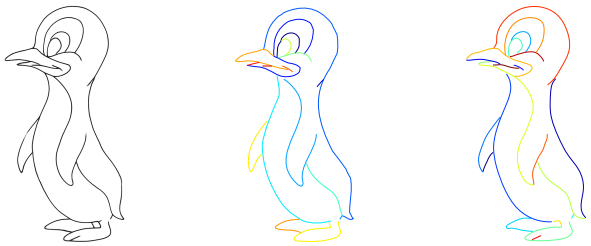
End-to-End Line Drawing Vectorization

Hanyuan Liu,[1] Chengze Li,[2] Xueting Liu,[2] Tien-Tsin Wong[1]*

[1] The Chinese University of Hong Kong
[2] Caritas Institute of Higher Education
liuhy@cse.cuhk.edu.hk, czli@cihe.edu.hk, tliu@cihe.edu.hk, ttwong@cse.cuhk.edu.hk

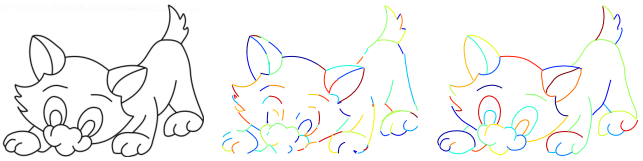
Abstract

Vector graphics is broadly used in a variety of forms, such as illustrations, logos, posters, billboards, and printed ads. Despite its broad use, many artists still prefer to draw with pen and paper, which leads to a high demand of converting raster designs into the vector form. In particular, line drawing is a primary art and attracts many research efforts in automatically converting raster line drawings to vector form. However, the existing methods generally adopt a two-step approach, stroke segmentation and vectorization. Without vector guidance, the raster-based stroke segmentation frequently obtains unsatisfying segmentation results, such as over-grouped strokes and broken strokes. In this paper, we make an attempt in proposing an end-to-end vectorization method which directly generates vectorized stroke primitives from raster line drawing in one step. We propose a Transformer-based framework to perform stroke tracing like human does in an automatic stroke-by-stroke way with a novel stroke feature representation and multi-modal supervision to achieve vectorization with high quality and fidelity. Qualitative and quantitative evaluations show that our method achieves state of the art performance.



(a) (b) (c)

Figure 1: (a) Input. (b) Guo et al. (c) Ours.



(a) (b) (c)

Figure 2: (a) Input. (b) Egiazarian et al. (c) Ours.

Introduction

Vector graphics play an important role in graphic design. The resolution independence and easy-for-editing features allow vector graphics to be used in a variety of forms, such as illustrations, logos, posters, billboards, and printed ads. Despite the broad use of vector graphics, many artists still prefer to draw with pen and paper which is more natural and easier to control. This leads to a high demand of converting raster designs into the vector form, especially for vectorizing line drawings which is a primary art form of graphic design. There exist many vector graphics editing softwares to help artists trace the vector lines from a raster line drawing, such as Adobe Illustrator and CorelDRAW, but the manual tracing process is still tedious and time-consuming.

Several research attempts have been made to automatically vectorize a raster line drawing. These methods generally face two key challenges in the line drawing vectorization task, stroke segmentation to identify individual strokes from the raster image and stroke vectorization to convert the identified strokes into the vectorized curve form. The existing methods generally tackle the above two challenges with a two-step approach, i.e., first identifying the strokes and then converting to the vector form. This highly limits the quality of stroke segmentation results without vector guidance. In particular, one stream of existing methods attempts to directly identify complete strokes via junction analysis (Guo et al. 2019), local neighborhood similarity (Kim et al. 2018), or region detection (Inoue and Yamasaki 2019), but these methods generally fail to separate strokes that locally similar and lead to over-grouped strokes (Fig. 1(b)). Another stream of methods first identifies smooth stroke segments and then connects the stroke segments into strokes based on local properties, such as proximity and continuity (Bessmeltsev and Solomon 2019; Egiazarian et al. 2020; Noris et al. 2013; Stanko et al. 2020; Mo et al. 2021). However, the quality of stroke connection generally cannot be guaranteed, leading to obvious broken stroke segments (Fig. 2(b)).

Different from the existing methods that use two-step approaches, in this paper, we propose an end-to-end vec-

*Corresponding author.

torization framework which directly outputs the vectorized stroke primitives from a raster line drawing, without using any post-processing or third-party tool for generating vector primitives. The key of our method is the integration of encoder-decoder-vectorizer models that can automatically generate all strokes in vector format and it allows our method to perform stroke tracing like a human does in an automatic stroke-by-stroke way. The encoder encodes the input line drawing into a list of stroke features whether each feature vector corresponds to a stroke. We explicitly encode the endpoint information in the stroke features so that the stroke features can be sorted and provides better hints for decoding and vectorization. The decoder decodes the stroke features into a raster reconstruction which provides supervision in raster form. The vectorizer decodes the stroke feature into vectorized strokes and provides supervision in vector form. With the multi-model loss in both vector and raster forms, we are able to vectorize the raster line drawing with high quality and fidelity.

We conduct comprehensive evaluations and ablation studies to validate the effectiveness of our method. Qualitative and quantitative evaluations show that our method achieves state of the art performance. The major contributions of our method can be concluded as follows:

- We propose an end-to-end vectorization method which directly generates vectorized stroke primitives from raster line drawing in one step.
- We propose a novel feature representation that encodes a line drawing into a sequence of sorted stroke features.
- We propose a multi-modal supervision that supervises our model in both raster and vector forms to achieve vectorization quality and fidelity.

Related Work

Line-Drawing Vectorization

Line-drawing vectorization aims to convert raster graphics into vector representations for simplification, compression, or resolution-free rendering purposes. Existing vectorization methods can be roughly classified into two branches: optimization-based and learning-based methods.

The optimization-based methods have been widely studied and are under active developments (Noris et al. 2013; Bessmeltsev and Solomon 2019; Stanko et al. 2020). These methods usually regard vectorization as an optimization process and attempt to minimize the cost of topology misalignment. However, these methods usually suffer from low computational efficiency and take a very long time to finish a single image. Recently, several learning-based solutions are proposed to tackle some subproblems in the vectorization pipeline with neural networks. These methods focus more on the problem and leverage the neural gradient descent to achieve a stroke segmentation or spline curve fitting in an automatic data-driven fashion. As a result, these methods are much faster in terms of vectorization inference. Among these methods, Kim et al. combine CNN and graph-cut algorithms to semantically segment the raster image into a set of paths. Inoue and Yamasaki formulate the line drawing vectorization as an instance segmentation problem and extend Mask R-CNN (He et al. 2020) for stroke extraction. Furthermore, Guo et al. takes one more step by processing the raster line drawing in a topology-aware manner. They use a CNN to identify the junctions and reconstruct the topology for each junction in pixel level. Afterward, they split the line drawing into individual curves and achieves vectorization by a least-square fitting. Here, we shall summarize the above methods as *pixel-based* solutions. These solutions extract strokes in the first place and implicitly achieves the vectorization as a side product with third-party tools such as Potrace (Selinger 2019). Therefore, the accumulated error in the model itself and the tracing algorithm may degrade the quality of generated primitives.

On the other hand, Gao et al., Egiazarian et al., and Mo et al. attempted to achieve direct vectorization without the intermediate step of stroke analysis or segmentation. To achieve so, Gao et al. use a hierarchical recurrent neural network to reconstruct the parametric spline curves and surfaces from raster images. Egiazarian et al. use a Transformer-based neural network to estimate the vector primitives from rasterized technical line drawings and cartoon line drawings. Mo et al. use a recurrent neural network supervised by a differentiable rasterization module (Li et al. 2020) to generate lines sequentially. Both (Gao et al. 2019), (Egiazarian et al. 2020), and (Mo et al. 2021) rely on the supervision on curve primitives, *i.e.primitive-level supervision*. However, these methods supervise the primitives of strokes independently without considering the semantics of the strokes, e.g., how each stroke contributes to the overall line drawing. As a result, such pure primitive level supervision may suffer from misalignment from raster lines and discontinuity between primitive segments.

Our work follows the idea of both streams of learning-based methods and leverages a combination of pixel-level supervision and primitive-level supervision: we implicitly segment each stroke from the raster input by pixel-level reconstruction errors and explicitly generate vector outputs by supervision on control points for each primitive.

Transformer

The Transformer architecture was first introduced by Vaswani et al. in 2017 as a new attention-based model architecture for machine translation tasks. It soon becomes the de facto standard building block for natural language processing tasks to now. Due to its superior representation power and affinity to sequential representations, it is not surprising to observe the wide usage of Transformers in computer vision tasks (Khan et al. 2021).

One noticeable work in directly applying Transformers to the vision task is the DETR model (Carion et al. 2020), which is proposed to detect the bounding boxes and predict the labels of objects from images in an end-to-end manner. Thanks to the help of the Transformer architecture, the DETR removes the need for conventional hand-designed components in object detection tasks and demonstrates improved accuracy and efficiency over other traditional detectors. More recently, Egiazarian et al. use a Transformer-based neural network to roughly estimate the vector primitives in their technical line drawing vectorization frame-

work. The predicted primitives are then refined by energy minimization. The results obtained in their two-step solution usually suffers from the discontinuity between primitive segments and missing semantics. In sharp contrast, our method bypasses any post-processing optimizations and can predict the primitives more accurately and directly. Our method supports direct image-to-vector translation in an end-to-end manner. Meanwhile, the auto-encoding design of our method ensures the efficiency and accuracy of our transcoding pipeline from raster to vector.

Method

Overview

Our proposed framework aims at an end-to-end vectorization of raster line drawings. Given a clean raster line drawing as input, the framework directly outputs a vector representation of the line drawing. We demonstrate an overview of the framework in Figure 3. As the figure indicates, the proposed framework consists of three modules, the *Stroke Encoder*, the *Stroke Decoder*, and the *Stroke Vectorizer*. Taking a line drawing image I as input, the *Stroke Encoder* first identifies the strokes in I and converts the strokes into an ordered list of stroke features $\{F_i\}$ in which each feature vector F_i represents a stroke of the raster input I. After the encoding, we propose the *Stroke Decoder* module and the *Stroke Vectorizer* module altogether to solve the stroke vectorization and the stroke reconstruction tasks simultaneously. Specifically, the *Stroke Vectorizer* takes the image I and the stroke feature F_i as input and estimates the control points $\{\theta_j\}_i$ for the stroke S_i. The *Stroke Decoder* decodes and reconstructs each stroke feature F_i and into raster stroke RS_i. Our framework relies on an auto-encoding scheme for a compact stroke representation. Each stroke S_i is encoded into a stroke feature F_i by minimizing the reconstruction errors. Moreover, the framework requires no tedious junction recognition nor any post-processing stages, which further simplifies the overall procedure of line drawing vectorization. We shall discuss the detailed design of each module in the following sections.

Stroke Encoder

First of all, we design the Stroke Encoder module to identify each stroke in the raster line drawing image input. The Stroke Encoder estimates the features of each stroke and outputs a list of feature vectors corresponding to the strokes. The Stroke Encoder features the Transformer architecture with a parallel encoding mechanism (Carion et al. 2020) in order to efficiently handle each stroke. Moreover, the Stroke Encoder is built to allow the processing of variable numbers of strokes.

To be specific, the module first encode the input raster image $I \in [0,1]^{W \times H}$ with a ResNet-based (He et al. 2016) feature extractor $X_f = \text{ResNet}(I)$, and then decode X_f with a sequence of n_{dec} Transformer blocks (Vaswani et al. 2017) to form our final list of feature embeddings $\{F_i\}$ as:

$$TransEmb = \text{Transformer}(PosEmb_{1d}, X_f), \quad (1)$$

$$\{F_i\} = \text{MLP}(TransEmb) \quad (2)$$

where $PosEmb$ denotes the positional encoding in the Transformer architecture and $TransEmb$ denotes the direct output embeddings from the transformer. The maximum number of strokes is set with the size of the input positional embedding $PosEmb_{1d} \in \mathbb{R}^{n_{stroke} \times d_{Trans}}$.

Comparing to existing solutions that directly use the Transformer outputs $TransEmb$ as the intermediate feature representation for downstream tasks such as object detection and object classification (Khan et al. 2021), we prefer to construct our intermediate feature list $\{F_i\}$ with a certain semantic meaning of the correlated stroke S_i. Based on our observation on the QuickDraw (Ha and Eck 2017) and the TU Berlin Sketch dataset (Eitz, Hays, and Alexa 2012) that the artists usually finish a line drawing stroke by stroke, we choose to explicitly constrain the encoded feature F_i to contain the starting point and an ending point of the stroke S_i. With the supervision of the endpoints, the model can better learn the morphology of each stroke in the line drawing. As a result, we first transform the $TransEmb$ by fully-connected network MLP and supervise the Transformer and the MLP to output the F_i as a sequence of $\{P_0, P_1, emb, p\}_i \in \mathbb{R}^{d_F = 5 + d_{emb}}$, where (P_0, P_1) represents endpoint coordinates of the stroke S_i. The latter emb in F_i is a transformed latent feature embedding of the stroke S_i to contain the essential information of the stroke for the following reconstruction and vectorization. Finally, p indicates the confidence value for the current estimation. If p is lower than 0.5, we shall simply remove F_i in our final list of feature embedding.

Feature Loss. We employ a feature loss to estimate the multi-task problem of primitive classification and control point supervision. Here we use a similar design of (Carlier et al. 2020; Egiazarian et al. 2020) to construct the feature loss as a composition of binary cross-entropy of the confidence and a weighted sum of L_1 and L_2 deviations of stroke endpoints as:

$$L_e\left(P_i, \hat{P}_i\right) = (1-\lambda_e)\left\|P_i - \hat{P}_i\right\|_2^2 + \lambda_e\left\|P_i - \hat{P}_i\right\|_1 \quad (3)$$

$$L_{confid}\left(p_i, \hat{p}_i\right) = -\hat{p}_i \log p_i - (1-\hat{p}_i)\log(1-p_i) \quad (4)$$

$$L_F = \frac{1}{n_{stroke}} \sum_{i=1}^{n_{stroke}} \left(\beta L_e + L_{confid}\right), \quad (5)$$

where $\beta = \min\left(ImgWidth, ImgHeight\right)$, $\lambda_e = 0.5$.

The target confidence values p_i are all ones, with zeros in the end indicating invalid placeholder strokes (Egiazarian et al. 2020). Since this loss function is not permutation-invariant with regard to the ordering of strokes and their endpoints, we sort the endpoints in each ground-truth strokes and the target stroke feature list by their endpoints lexicographically.

Stroke Decoder

In the previous step, we encode the strokes of the input image I to a list of features F_i with the Stroke Encoder Module. However, the Stroke Encoder only constrains the endpoint location and leaves the feature embedding emb of F_i unconstrained. To constrain the emb of each feature entry F_i to

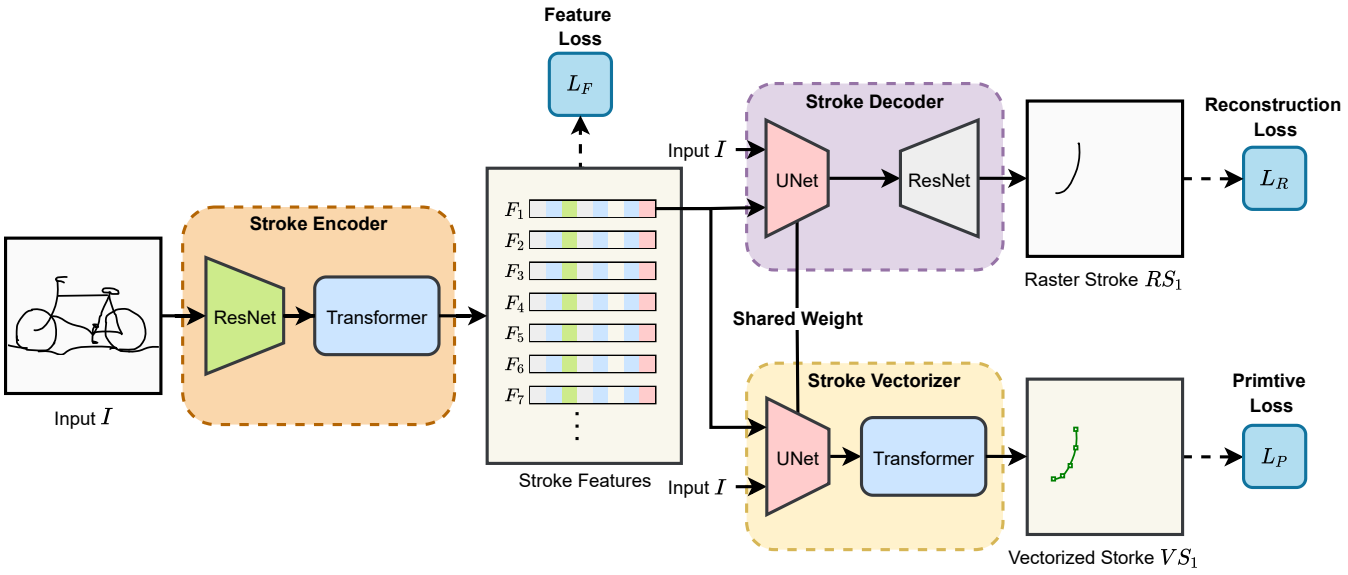


Figure 3: Overview. Given raster image I, the Stroke Encoder identifies strokes and encodes them into stroke features $\{F_i\}$; the Stroke Decoder decodes stroke feature F_i into raster stroke RS_i; the Stroke Vectorizer generates the control points for vector stroke VS_i based on the stroke feature F_i and the input image I.

be informative about the stroke S_i, we introduce the Stroke Decoder module. The module processes strokes individually and enforces accurate reconstruction of the raster representation of S_i from the emb. With this pixel-level stroke supervision, we ensure the stroke feature F_i contains necessary information and further benefits the following Stroke Vectorizer Module to give more accurate results.

To achieve so, we first propose a U-Net based backbone network (Ronneberger, Fischer, and Brox 2015) denoted as UNet to give a feature transformation of encoded stroke feature F_i of stroke S_i, conditioned by the input image I, as $X_s = \text{UNet}(F_i, I)$. The backbone network takes I as input. F_i is concatenated in the innermost layer of the backbone network. To improve the model performance, a mesh grid composed of the coordinates of each pixel (Liu et al. 2018) is also passed into the backbone network UNet. The transformed feature X_s is afterward fed into a ResNet-based (He et al. 2016) convolutional neural network to be decoded to reconstruct the raster stroke RS_i. After all, we combine the U-Net backbone network and the ResNet-based image reconstructor to form the complete Stroke Encoder module.

Stroke Reconstruction Loss. As discussed above, the Stroke Decoder focuses on the reconstruction quality of the raster strokes RS_i to ensure the stroke feature F_i (especially the emb portion) to be meaningful. Thus, we use a combination of L_1 and L_2 loss to calculate the pixel-wise difference between the ground-truth stroke GT_S_i and the reconstructed raster stroke RS_i. We hence formulate the stroke reconstruction loss as below:

$$L_R\left(S_{gt}, RS\right) = (1-\lambda_r)\left\|S_{gt} - RS\right\|_2^2 + \lambda_r \left\|S_{gt} - RS\right\|_1 \quad (6)$$

where $\lambda_r = 0.3$.

Stroke Vectorizer

Given an extracted feature representation F_i of each stroke, we formulate the stroke vectorization as an auto-regressive prediction problem, i.e., given feature F_i and a starting point, we progressively output one segment of the vector stroke primitives at a time, until all primitives of the strokes are all outputted. Significantly, the output of the n-th primitive is both conditioned by the feature F_i and all $n-1$ previously outputted strokes. This auto-regressive prediction process imitates the actual pen-ink tracing of human artists, in which new lines or curves (Beziér curve in our case) are drawn continuously along the original stroke. Moreover, the auto-regressive manner always outputs sequentially connected and naturally ordered vector primitives so as to improve the accuracy and the representation efficiency of the vectorization.

Here, we propose the Stroke Vectorizer module to solve this auto-regressive tracing problem. The Vectorizer starts with the same feature transformation $X_s = \text{UNet}(F_i, I)$ in the Stroke Decoder. After that, we realize the auto-regressive tracing of the stroke with a Transformer network (Vaswani et al. 2017). The transformer network reads X_s and predicts a list of Beziér curve primitives lst as the final output for the i-th stroke.

For a single stroke, we start at the endpoint (x_s, y_s) that is of closer coordinate location to 0. For simplicity, we use a partial format $curve_t = (x_{c1}, y_{c1}, x_{c2}, y_{c2}, x_e, y_e)_t$ to represent a Beziér curve instead. In this format, we can ensure the stroke vector continuity because the starting point of $curve_t$ is also the ending point of $curve_{t-1}$. To bootstrap the auto-regressive prediction, we manually specify the initial curve primitive as $curve_0 = (x_s, y_s, x_s, y_s, x_s, y_s)$. Given

Algorithm 1: Stroke Vectorizer

Input: $curve_0$, feature map X_s
Result: lst
Init $lst_0 = \{curve_0\}$;
Init eos = false;
while *not* eos **do**
  $curve_t$, eos = Transformer(lst_{t-1}, X_s) ;
  $lst_t = lst_{t-1} + curve_t$
end

the transformed feature X_s and the initial set of curves lst_0, we describe the actual tracing algorithm with Transformers in Algorithm 1. Especially, we follow the original Transformer design to use a multi-Layer perceptron for predicting the next primitive $curve_t$ and the End of Sequence symbol eos. The eos symbol indicates no remaining Beziér primitives to be generated.

Primitive Loss. Despite the auto-regressive mechanism of the Stroke Vectorizer, we can simply train it with a multi-task primitive Loss. The primitive loss is similar to the Feature Loss in Eq. 5 except the primitives are ordered naturally. The loss consists of a binary entropy loss for eos and an affine combination of L_1 and L_2 loss for the control points of each curve:

$$L_{eos}\left(e_i, \widehat{e}_i\right) = -\widehat{e}_i \log e_i - (1-\widehat{e}_i)\log(1-e_i) \tag{7}$$

$$L_c\left(\theta_i, \hat{\theta}_i\right) = (1-\lambda_c)\left\|\theta_i - \hat{\theta}_i\right\|_2^2 + \lambda_c\left\|\theta_i - \hat{\theta}_i\right\|_1 \tag{8}$$

$$L_P = \frac{1}{n_{primitive}} \sum_{i=1}^{n_{primitive}} (\beta L_c + L_{eos}) \tag{9}$$

where $\beta = \min(ImgWidth, ImgHeight)$, $\lambda_c = 0.75$.

Overall Loss Function

In summary, our multi-task training loss function is made up of three components: (1) a feature loss L_F for stroke feature estimation, (2) a raster stroke reconstruction loss L_R for pixel-level supervision, and (3) a primitives prediction loss L_P that enables vectorization. The total loss is formulated as follows:

$$L_{total} = L_F + \frac{1}{n_{stroke}}(\lambda_R L_R + \lambda_P L_P) \tag{10}$$

where n_{stroke} is the number of the strokes in the input image, λ_R and λ_P are 0.1 and 6.0 respectively.

Experiments

Implementation Details

Dataset Our framework is able to process images with arbitrary resolution. Due to the hardware memory limit and the efficiency considerations, we train our network with a relatively lower resolution. In the evaluation phase, we allow higher-resolution inputs. We use a portion (excluding the BACKPACK and the BICYCLE classes for validation) of QuickDraw (Ha and Eck 2017) and the TU Berlin (Eitz, Hays, and Alexa 2012) dataset for the framework training. We render the vector data as raster images of varying resolutions from 64 px to 256 px with a step size of 32 px.

To evaluate the performance of our framework, we perform both quantitative and qualitative comparisons. We use 1150 images from the testing dataset provided by (Kim et al. 2018) (selected images from the BACKPACK and the BICYCLE classes on the QuickDraw dataset) for quantitative evaluation. For the qualitative comparison, we first randomly sample drawings in the validation dataset. Moreover, to validate the robustness of our model, we also test with several line drawings in previous works including (Egiazarian et al. 2020; Guo et al. 2019) in our qualitative comparison.

Implementation We implement our framework in PyTorch following a similar Transformer design as (Carion et al. 2020; Egiazarian et al. 2020) with 8 decoder layers for Stroke Encoder and 6 encoder/decoder layers for Stroke Vectorizer. Both of the Transformers use 8 attention heads. For Stroke Encoder, the 1D sinusoidal positional encoding is used for parallel decoding (Carion et al. 2020). We use three independent Adam optimizers for Stroke Encoder, Stroke Decoder, and Stroke Vectorizer, all with an initial learning rate of 0.0001. We trained our model using 4 NVIDIA Titan V GPUs with automatic mixed precision, gradient accumulation trick, and gradient checkpointing (Chen et al. 2016) through all our experiments.

More technical information, such as the detailed network architecture and the interconnect between each component, will be included in the supplementary material.

Training Our framework is sensitive to initialization due to unbalanced learning loads between the Stroke Extractor and the Stroke Decoder/Vectorizer (as an input image can contain several strokes). The joint optimization of all three modules with randomly initialized weights might not converge. To ensure the training stability of our framework, we adopted a *bootstrap* training scheme. We first randomly pick 5,000 raster input training images as a bootstrap collection $\{I_i\}_{bs}$. We assign a learnable embedding $\widehat{emb}_{i,j}$ for each stroke $S_{i,j}$ belonging to the bootstrap collection. We then jointly optimize the $\widehat{emb}_{i,j}$ and the Stroke Decoder with the ground truth endpoints $P_{i,j}$ and raster stroke $RS_{i,j}$ until convergence. Stroke Encoder is then directly supervised with $(I_i, \{P_{i,j}\}, \{\widehat{emb}_{i,j}\})$. Stroke Vectorizer is also trained with the $\widehat{emb}_{i,j}$ and primitives of the related vector stroke. In other words, we leverage the Stroke Decoder as a gradient-based solver for embedding probe. After the *bootstrap* stage, we jointly train the whole framework with all training data until convergence.

Evaluations

Visual Comparisons Figure 4 shows visual comparisons between our framework and the state-of-the-art methods on low-resolution input. The predicted strokes are visualized with distinct colors. The results of (Kim et al.), (Inoue and Yamasaki), and (Mo et al.) contain both over-grouped and unconnected strokes. (Egiazarian et al.) generates massive

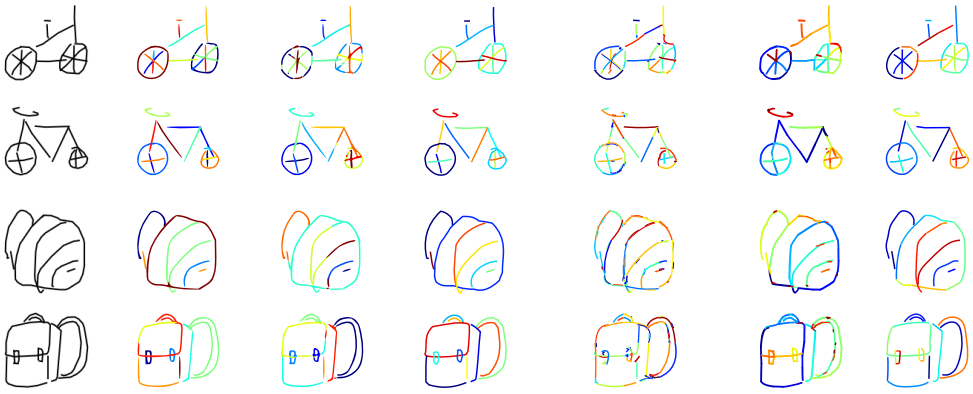


Figure 4: Vectorization results generated by different methods. The resolution of all input images is 128×128.

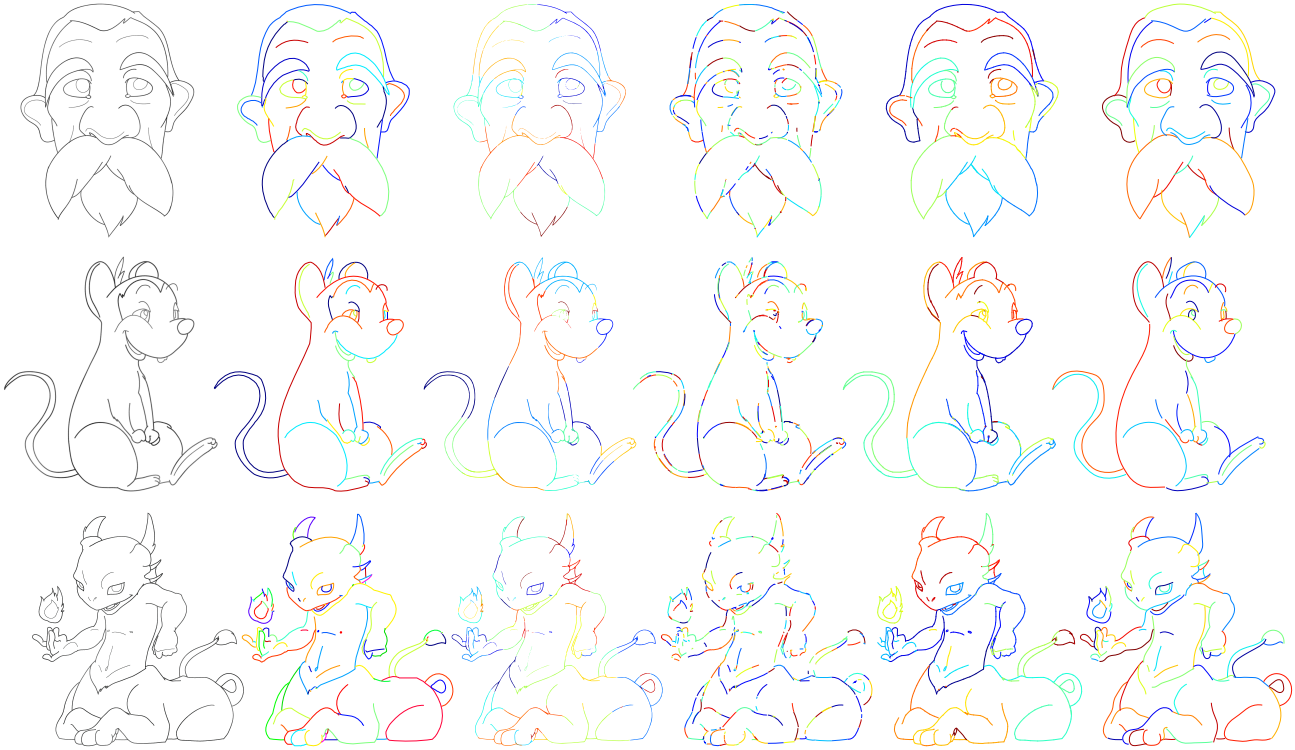


Figure 5: Vectorization results on high-resolution line drawings. The resolution of all input images is 1024×1024.

unconnected strokes. Both (Guo et al.) and our method show visually pleasant results.

Figure 5 shows visual comparisons on high-resolution input. We exclude (Kim et al. 2018) from this comparison, since its optimization process takes more than one day to handle a high-resolution input. Both (Inoue and Yamasaki) and (Egiazarian et al.) generate massive unconnected strokes. (Mo et al.) also generates many unconnected strokes and misses some details. (Guo et al.) contains several over-grouped strokes in the final output. Though our method also degrades a little bit for high-resolution input, we can still obtain satisfying results for all examples.

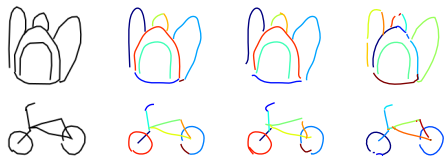
Quantitative Comparisons Since there are no direct evaluation metrics for the estimated primitives, we can only measure the pixel-level similarity between the input line drawing and the rasterized output using five similarity metrics, i.e., Structural similarity index (SSIM), Intersection-

	SSIM ↑	IoU ↑	HD ↓	CD ↓	EMD ↓
Kim et al.	0.9764	56.5%	50.6	0.0298	0.516
Inoue and Yamasaki	0.9758	54.8%	**49.8**	0.0413	0.575
Guo et al.	**0.9788**	**60.6%**	50.9	<u>0.0276</u>	<u>0.444</u>
Egiazarian et al.	0.8914	42.1%	51.2	0.778	2.108
Mo et al.	0.9505	51.7%	51.6	0.0286	0.445
Ours	<u>0.9785</u>	<u>58.3%</u>	<u>50.3</u>	**0.0143**	**0.403**

Table 1: Quantitative evaluation of learning-based vectorization methods. The best and second best results are in bold and underline font respectively.

	Full	W/o Pixel	W/o Primitive
SSIM↑	**0.9785**	0.9456	0.9634
IoU↑	**58.3%**	41.0%	49.7%
HD↓	**50.3**	51.3	50.7
CD↓	**0.0143**	0.0872	0.0556
EMD↓	**0.403**	0.921	0.601

Table 2: Ablation study on different loss terms.



(a) (b) (c) (d)

Figure 6: Ablation study of our framework. (a) Input. (b) Full supervision. (c) W/o pixel-level supervision. (d) W/o primitive-level supervision.

over-Union (IoU), Hausdorff distance (HD), Chamfer distance (CD), and Earth mover's distance (EMD). Here, while IoU can reflect deviations in two rasterized vector graphics, it cannot capture the spatial deviations between raster shapes, *e.g.* two similar shapes with minor offsets from each other. Thus, we introduce distance metrics for shape evaluation. We treat the rasterized images as point clouds and calculate the distance metrics on these point clouds.

The statistics are shown in Table 1. We want to emphasize that since the evaluations are performed on pixels instead of primitives, the similarity metrics can only measure the fidelity of the generated vector graphics, but fail to capture the overgrouping or undergrouping of strokes. In terms of fidelity, our model significantly outperforms (Egiazarian et al.), slightly outperforms (Inoue and Yamasaki), (Kim et al.) and (Mo et al.), and shows comparable performance with (Guo et al.).

Ablation Study To verify the effectiveness of our framework design, we conduct ablation studies on different loss terms. The evaluation results are shown in Table 2.

Feature Supervision. We explicitly supervise the stroke endpoints and stroke confidence of the stroke feature list in Stroke Encoder. We found this supervision is critical for our whole framework. We trained the whole framework without feature supervision and found it does not converge.

Pixel-Level Supervision. As shown in Figure 6 (c), the framework trained without pixel-level supervision generates low-quality vector outputs. The primitives show large deviations and are misaligned from their raster counterparts. The quantitative metrics are worst compared with others.

Primitive-Level Supervision. Without the primitive supervision, our model is incapable of producing the vectorization results. Therefore, We trace the reconstructed raster strokes with Potrace, and use the traced results for evaluation. As shown in Figure 6 (d), the strokes are broken apart, which leads to visually-unpleasing results.

Limitation and Discussion

Currently, our framework only works for clean line drawing. It cannot process messy line drawings and rough sketches. We require an explicit simplification prior to our framework for these kinds of inputs. On the other hand, our framework use Transformer to understand and translate the sequential stroke information. Due to the extensive self-attention computation, the computational cost of our framework is relatively higher. As a result, our model cannot directly process high-resolution images on current commodity GPUs (we instead evaluate high-resolution images on CPU). We may investigate the possibilities to include efficient attention mechanisms (Child et al. 2019; Niculae and Blondel 2017) for future improvements.

Conclusions

We proposed an end-to-end vectorization framework to cope with the challenging free-form line drawing vectorization task in this work. Unlike existing solutions, our method directly outputs vector representation from line drawing strokes without extra post-processing or tracing steps. The major advantage of our framework comes firstly from the auto-encoding scheme of the Stroke Encoder and the Stroke Decoder to enrich the representation ability of the stroke vectors. Moreover, with the auto-regressive manner of the stroke vectorizer, we enable end-to-end outputs from raster to vector stroke primitives. Both qualitative and quantitative supports that our framework achieves state-of-the-art performance among existing learning-based vectorization methods.

Acknowledgements

This project is supported by CUHK Direct Grant for Research (Project No. 4055152).

References

Bessmeltsev, M.; and Solomon, J. 2019. Vectorization of Line Drawings via Polyvector Fields. *ACM Trans. Graph.*, 38(1).

Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; and Zagoruyko, S. 2020. End-to-End Object Detection with Transformers. In *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, 213–229. Cham: Springer International Publishing. ISBN 9783030584528.

Carlier, A.; Danelljan, M.; Alahi, A.; and Timofte, R. 2020. DeepSVG: A Hierarchical Generative Network for Vector Graphics Animation. In *Advances in Neural Information Processing Systems*, volume 33, 16351–16361. Curran Associates, Inc.

Chen, T.; Xu, B.; Zhang, C.; and Guestrin, C. 2016. Training Deep Nets with Sublinear Memory Cost. arXiv:1604.06174.

Child, R.; Gray, S.; Radford, A.; and Sutskever, I. 2019. Generating Long Sequences with Sparse Transformers. arXiv:1904.10509.

Egiazarian, V.; Voynov, O.; Artemov, A.; Volkhonskiy, D.; Safin, A.; Taktasheva, M.; Zorin, D.; and Burnaev, E. 2020. Deep Vectorization of Technical Drawings. In *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, 582–598. Cham: Springer International Publishing. ISBN 9783030586010.

Eitz, M.; Hays, J.; and Alexa, M. 2012. How Do Humans Sketch Objects? *ACM Trans. Graph.*, 31(4).

Gao, J.; Tang, C.; Ganapathi-Subramanian, V.; Huang, J.; Su, H.; and Guibas, L. J. 2019. DeepSpline: Data-Driven Reconstruction of Parametric Curves and Surfaces. arXiv:1901.03781.

Guo, Y.; Zhang, Z.; Han, C.; Hu, W.; Li, C.; and Wong, T.-T. 2019. Deep Line Drawing Vectorization via Line Subdivision and Topology Reconstruction. *Computer Graphics Forum*, 38(7): 81–90.

Ha, D.; and Eck, D. 2017. A Neural Representation of Sketch Drawings. arXiv:1704.03477.

He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R. 2020. Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2): 386–397.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. Los Alamitos, CA, USA: IEEE Computer Society.

Inoue, N.; and Yamasaki, T. 2019. Fast Instance Segmentation for Line Drawing Vectorization. In *2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM)*, 262–265.

Khan, S.; Naseer, M.; Hayat, M.; Zamir, S. W.; Khan, F. S.; and Shah, M. 2021. Transformers in Vision: A Survey. arXiv:2101.01169.

Kim, B.; Wang, O.; Öztireli, A. C.; and Gross, M. 2018. Semantic Segmentation for Line Drawing Vectorization Using Neural Networks. *Computer Graphics Forum*, 37(2): 329–338.

Li, T.-M.; Lukáč, M.; Gharbi, M.; and Ragan-Kelley, J. 2020. Differentiable Vector Graphics Rasterization for Editing and Learning. *ACM Trans. Graph.*, 39(6).

Liu, R.; Lehman, J.; Molino, P.; Petroski Such, F.; Frank, E.; Sergeev, A.; and Yosinski, J. 2018. An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution. In *Advances in Neural Information Processing Systems*.

Mo, H.; Simo-Serra, E.; Gao, C.; Zou, C.; and Wang, R. 2021. General Virtual Sketching Framework for Vector Line Art. *ACM Trans. Graph.*, 40(4).

Niculae, V.; and Blondel, M. 2017. A Regularized Framework for Sparse and Structured Neural Attention. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 3340–3350. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781510860964.

Noris, G.; Hornung, A.; Sumner, R. W.; Simmons, M.; and Gross, M. 2013. Topology-Driven Vectorization of Clean Line Drawings. *ACM Trans. Graph.*, 32(1).

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Lecture Notes in Computer Science, 234–241. Cham: Springer International Publishing. ISBN 9783319245744.

Selinger, P. 2019. Potrace. http://potrace.sourceforge.net. Accessed: 2021-09-01.

Stanko, T.; Bessmeltsev, M.; Bommes, D.; and Bousseau, A. 2020. Integer-Grid Sketch Simplification and Vectorization. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Geometry Processing)*, 39(5): 149–161.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.