Semantically Meaningful Vectorization of Line Art in Drawn Animation

Calvin Metzger[*]
Supervised by: Univ.Prof. Michael Wimmer [†]

Faculty of Informatics
TU Wien
Vienna / Austria

Abstract

Animation consists of sequentially showing multiple single frames with small mutual differences in order to achieve the visual effect of a moving scene. In limited animation, these frames are drawn as semantically meaningful vector images which could be referred to as clean animation frames. There are limited animation workflows in which these clean animation frames are only available in raster format, requiring laborious manual vectorization.

This work explores the extent to which line-art image vectorization methods can be used to automatize this process. For this purpose, a line-art image vectorization method is designed by taking into account the structural information about clean animation frames. Together with existing state-of-the-art line-art image vectorization methods, this method is evaluated on a dataset consisting of clean animation frames. The reproducible evaluation shows that the performance of the developed method is remarkably stable across different input image resolution sizes and binarized or non-binarized versions of input images, even outperforming state-of-the-art methods at input images of the default clean animation frame resolution. Furthermore, it is up to 4.5 times faster than the second-fastest deep learning-based method. However, ultimately the evaluation shows that neither the developed method nor existing state-of-the-art methods can produce vector images that achieve both visual similarity and sufficiently semantically correct vector structures.

Keywords: vectorization, line-art, animation, deep-learning

1 Introduction

In principle, animation consists of sequentially showing single frames in order to achieve the visual effect of a moving scene. *Limited animation* is an animation technique in which frames are not completely redrawn (like in full animation), but where the moving parts (also called *cels*) are reused over frames.

The hand-drawn limited-animation production process is composed of four phases. Based on the storyboard produced in the first phase, animators repeatedly draw and improve rough key frames in the second phase. These keyframes are line drawings only drawn for critical moments in a scene and contain mostly cels. In the third phase, the rough keyframes cleaned of any spurious lines or obsolete text markers and vectorized. To achieve the visual effect of fluidity, a large number of frames in between the keyframes are drawn. Finally, in the fourth phase, the clean frames are colored and enriched with special effects and a background image.

In order for the limited animation production process to proceed as quickly and as accurately as possible, clean frames need to be drawn as vector images. In the event of clean animation frames being only available in raster format, it is necessary to manually vectorize the images before they can be used efficiently. Automatizing this process is challenging, as the resulting line-art vector image needs to be semantically meaningful, i.e., the arrangement, topology and parameterization of graphical primitives (i.e., Bézier curves) need to make sense and be close to how artists would draw. An example of such a process is depicted in Figure 1

In order to alleviate this issue, this work will attempt to answer the **Research Question 1 (RQ1)**: To what extent is it possible to automatically vectorize clean animation frame line art in a manner that is semantically meaningful?

To answer RQ1, the Research Objective 1 (RO1) is to create a method for line-art vectorization that takes clean animation frame raster images as input and outputs the corresponding semantically meaningful vector image. This method is based on a deep learning model tailored to the qualitative structure of clean animation frames as input and output images, as traditional heuristics-based algorithms [15, 19, 11] tend to produce vector images that visually resemble the raster image closely, but contain semantically meaningless vector primitives.

Accordingly, the Research Objective 2 (RO2) is to perform an evaluation that ascertains the extent to which the developed method and existing state-of-the-art line-art image

[*]calvin.metzger@student.tuwien.ac.at
[†]wimmer@cg.tuwien.ac.at

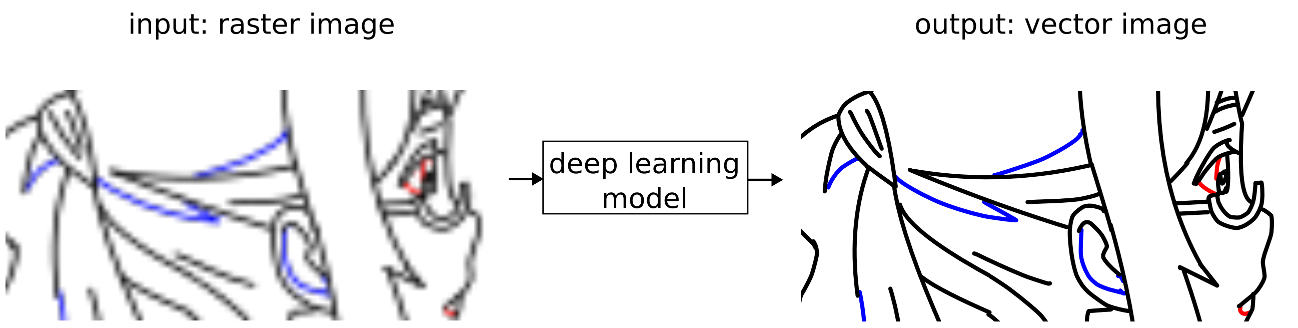


Figure 1: Overview of the research objective. The objective is to automatically convert clean animation frame line-art raster images into vector images. Zooming into the figure reveals the structural difference between the input and the output image. Input and output images are provided by Tonari Animation. Note that the output image is taken from the gold standard test dataset. For a genuine reconstruction result of the developed line-art vectorization method, refer to Figure 5.

vectorization methods are able to vectorize clean animation frames.

The code for this work is publicly available at `https://github.com/nopperl/marked-lineart-vectorization`.

2 Related Work

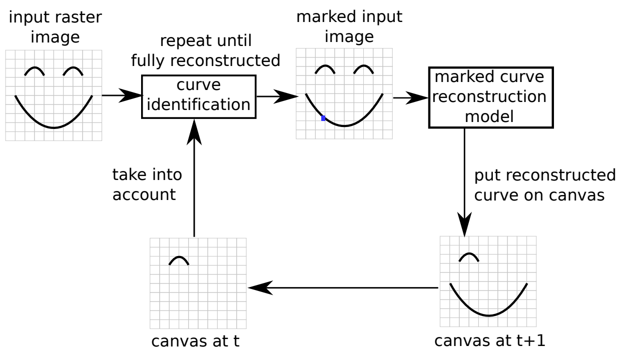
This section details existing work on image vectorization, specifically for the case of line art. Since there is a non-injective relation between vector images and raster images, converting a raster image into a vector image is a non-trivial task. Hence, state-of-the-art methods primarily utilize learned models to achieve this. While there exist methods based solely on heuristic optimization [15, 19, 11, 1, 21], they do not produce the intended output for this task, as the resulting vector primitives rarely resemble the primitives an artist would draw. Additionally, they require manual hyperparameter tuning for each individual image. Furthermore, each method relies on strong assumptions on the input image, such as exceeding a specific resolution, a low signal-to-noise ratio or containing only specific junctions.

While image vectorization is not yet a solved task, there have been some recent advances in deep learning for vector images. Reddy [13] introduce Im2Vec, an encoder-decoder architecture consisting of a Convolutional Neural Network (CNN) encoder and a Recurrent Neural Network (RNN) decoder. The CNN encodes the image into a latent feature vector, while the RNN is used to decode this feature vector into a fixed-length sequence of vector shapes based on multiple Bézier curves. It can be trained to vectorize raster images without vector supervision. This would be very useful in the context of line-art vectorization. The ability to train the model without vector supervision stems from its usage of a differentiable rasterizer [8]. In the general case, there are two main limitations of Im2Vec: The pixel resolution has to be defined at training time and the model does not scale well to higher resolutions. Additionally, the outputs sometimes contain degenerate features or semantically useless parts. Furthermore, Im2Vec only works on a specific type of image, such as emojis or icons. Finally, there were no experiments in the paper to output more than 4 shapes. Hence, it is doubtful whether it is possible to train the RNN decoder to output the large number of Bézier curves required for a clean animation frame.

The virtual sketching framework introduced by Mo et al. [10] is similar to Im2Vec in that it is trained without vector supervision to vectorize raster images. Other than that, it differs from Im2Vec in multiple ways. The main difference is that it constrains the output to only produce quadratic Bézier curves. Also, it is an iterative model, i.e. the curves are sequentially added to a canvas in a differentiable manner. After a given number of curves is drawn, the loss is computed and propagated through all the steps. These two differences make the model more suitable for professional line art. However, since the iterative model is trained mainly by computing a perceptual loss [6] of the whole output image with the input image, the results are not semantically meaningful vector images.

A different approach is to incorporate parts of traditional optimization-based methods. Based on earlier work [1], the state-of-the-art method by Puhachov et al. [12] uses a learned ensemble model to detect curve keypoints (such as junctions, start/end points and corners). Together with the input image, these keypoints are used by a geometric flow algorithm to find connections between keypoints and compute their geometry. It achieves remarkably good results, but has a more narrow aim than the proposed work. The algorithm focuses on retaining the correct stroke connectivity in the presence of noise, in their case for scanned pencil drawings. However, clean animation frames are not noisy and the curves are more narrow and densely connected, forming one large connected component for curves.

On the other hand, there do exist works that attempt to fully learn a line-art vectorization model using (partially) vector supervision, which makes it easier to produce semantically meaningful vector images [18, 4, 2]. Of note is a method to generate technical drawings by Egiazarian et al. [3]. It uses the Transformer [16, 14] architecture and is constricted to only handle 10 curves per image. To handle


input raster image
repeat until fully reconstructed
curve identification
marked input image
marked curve reconstruction model
take into account
put reconstructed curve on canvas
canvas at t
canvas at t+1

(a) The method unrolled at time step $t+1$

Figure 2: Overview of the proposed method. The method iteratively reconstructs a given raster line-art image as a vector image. At time step $t=0$, an algorithm identifies a new curve to reconstruct and places a marker on it. This information is then passed to a learned marked-curve reconstruction model to reconstruct the curve in vector format using cubic Bézier curve parameters. This output is added to a canvas, which is taken into account when identifying the curve to reconstruct at $t+1$.

images with a larger amount of curves, each image is split into fixed-size tiles. The tiles are processed independently by using the Transformer model to predict vector primitives to match the curves in the image. The resulting primitives are then refined using a physics-inspired algorithm by aligning them to the black pixels in the raster image. Afterwards the primitives of all tiles are merged using a simple heuristic algorithm. While the model produces good results on technical line drawings, the authors also demonstrate that it generalizes to other line art. It is limited by the assumption that there are less than 10 curves within a tile and the reliance on the heuristic merging algorithm.

3 Method

This section describes a method to automatically convert line-art raster images into vector images. The method is visualized in Figure 2 and consists of two parts: the main part is a learned model that takes as input a raster line-art image and a mark on a curve in this image and outputs a cubic Bézier curve which fits the marked curve, which is described in Section 3.1. The second part is a lightweight algorithm that uses this model iteratively to reconstruct all curves in an image, which is described in Section 3.2.

The method is designed in an iterative manner in order to handle the large amount of Bézier curves in the considered line-art images. Additionally, this structure is more amenable to manual fixing of the output, since missing curves can easily be reconstructed by invoking the curve reconstruction part with a marker on the curve in question.

3.1 Marked-Curve Reconstruction Model

The marked-curve reconstruction model architecture is depicted in Figure 3 and was designed by following the principle that reducing the complexity of the task the model needs to solve increases the probability that the model actually converges to a suitable state.

This is achieved by three design decisions. The most important design decision is to have the model reconstruct only a *single* curve instead of all curves per invocation. The other two decisions are based on the input and the output of the model and are explained below.

3.1.1 Input and Output

The input of the model is a line-art raster image. Additionally, this image contains one marker pixel placed on a curve to reconstruct. Importantly, this means that the *location* of the curve is already established. This information can be used to reduce the task complexity for the model by centering the input image on the mark.

The raster input images are represented using the Red-Green-Blue (RGB) color model, i.e., each pixel is represented using three floating-point numbers in $[0,1]$.

The output of the model is defined as the parameters of a cubic Bézier curve with a fixed stroke width. The parameters are defined by the start point, the end point and two control points, resulting in a vector of length 8. This output structure is sufficient to represent the output data domain considered in this work, i.e., clean animation frames.

3.1.2 Model Architecture

The architecture of the marked-curve reconstruction model is depicted in Figure 3. It consists of an encoder neural network that turns the input image $\mathbf{x}$ into a latent vector $\mathbf{z}$ of length L, and a decoder neural network that turns this latent vector into cubic Bézier curve parameters $\mathbf{o}$.

Since the input is an image, the encoder is a convolutional neural network. A global average pooling layer [9] is used at the end to produce a latent vector of predefined length L independent of the input image size. The hyperparameters of the encoder layers are displayed in Table 1.

Note that, as described above, the encoder architecture is designed to handle variably sized input, with these variables being denoted in Table 1. The batch size B is used to process multiple observations in parallel and increase the effectiveness of batch normalization by decreasing the variance. The image width W and height H need to be a multiple of 2, but can be otherwise freely chosen. The latent vector length L needs to correspond to the length used for the input vector of the decoder. For this work, the hyperparameters are set to $B=32$, $W=H=512$ and $L=128$. $L=128$ was chosen after early experiments with smaller resolutions. W and H are set to a square multiple of 2 approaching the maximum clean animation frame resolution

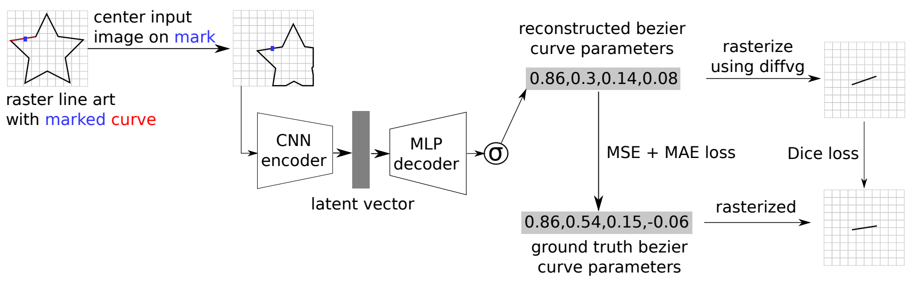

center input image on mark
raster line art with marked curve
CNN encoder
latent vector
MLP decoder
σ
reconstructed bezier curve parameters
0.86,0.3,0.14,0.08
MSE + MAE loss
0.86,0.54,0.15,-0.06
ground truth bezier curve parameters
rasterize using diffvg
Dice loss
rasterized

Figure 3: Architecture overview of the marked-curve reconstruction model. Note that for brevity, lines with two points are shown instead of cubic Bézier curves with four points.

of 1280×720 pixels. Note that the width and height deliberately do not correspond to the exact resolution of clean animation frames in the dataset to show that the model does not overfit to a specific resolution. B is maximized under the constraint of a limited amount of Graphics Processing Unit (GPU) memory.

The decoder is summarized in Table 2 and is a 2-layer multi-layered perceptron (MLP), which turns the latent vector of length L into a vector of length $P*2$, where P is the number of cubic Bézier curve parameters. Since cubic Bézier curves are parameterized by a start point, an end point and two control points, $P = 4$. Hence, the output is restricted to $[0,1]$ using the sigmoid function. The x-coordinates of the cubic Bézier curve points are then scaled with the image width, while the y-coordinates are scaled with the image height.

3.1.3 Training

The model is trained using supervised learning with a combination of a raster-based loss for visual similarity and a vector-based loss for semantic correctness. This task-derived loss combination is an important distinction from related work [13, 10, 3].

The vector loss follows Egiazarian et al. [3] and is an even combination of mean absolute error (MAE) and mean squared error (MSE). Defining a raster-based loss is more difficult, since the model outputs the cubic Bézier curve in vector format, which needs to be rasterized in a differentiable manner. The differentiable rasterizer introduced by Li et al. [8] is used for this. The raster output image is then compared to the rasterized ground truth image, with all curves aside from the marked curve removed. Using this, the dice loss function in Equation (1) can be used as loss. Note that o is the model output and y is the ground truth raster image.

$$\text{dice}(o,y) = 1 - \frac{2yo+1}{y+o+1} \tag{1}$$

The model is trained using the widely used Adam [7] optimizer with a learning rate of $\eta = 5*10^{-4}$.

3.2 Iterative Curve Reconstruction Algorithm

The marked-curve reconstruction model introduced in Section 3.1 is the main part of the line-art image vectorization method, but reconstructs only a single curve without color or stroke width information given a marked curve on the line-art raster image. In order to vectorize an entire line-art raster image, an algorithm has to be defined around the model that performs three tasks detailed in the following sections.

3.2.1 Color and Stroke Width

For the first task, recall that the marked-curve reconstruction model does not output color information. Since color carriers significant meaning in clean frames, it is necessary for the algorithm to produce the correct color information for all predicted curves.

This can easily be done for clean animation frames as they are drawn according to a color scheme which is known a priori. Hence, the image can be simply segmented according to these colors. For the dataset considered in this paper, these segments already exist. Then, the curve colors of each segment are set to black and each segment is individually input into the marked-curve reconstruction model. The color of its output can then be set to the segment color.

In the same vein, the marked reconstruction model does not output stroke width information. However, in clean animation frames, all curves share the same stroke width by design. Hence, it is possible to assume a constant stroke width for the input image and to apply it to all reconstructed curves.

3.2.2 Curve Identification

In order to indicate to the marked-curve reconstruction model which curve needs to be reconstructed, the second task consists of sampling a pixel lying on a curve not already reconstructed given the input image (more specifically an input image segment, as described in Section 3.2.1) and a canvas image containing already reconstructed curves. In the case of clean line-art images considered in this work, this can simply be done by sampling a random black pixel.

layer	output shape	# params	filter size	kernel size	stride	padding
2-d conv	$(B, 32, W/2, H/2)$	896	32	3	2	1
2-d conv	$(B, 64, W/4, H/4)$	18496	64	3	2	1
2-d conv	$(B, 128, W/8, H/8)$	73856	128	3	2	1
2-d conv	$(B, 256, W/16, H/16)$	295168	256	3	2	1
2-d conv	$(B, 512, W/32, H/32)$	1180160	512	3	2	1
2-d conv	$(B, L, W/32, H/32)$	589952	L	3	1	1
avg pool + squeeze	(B, L)	0	L	$W/32$	-	-

Table 1: Summary of the layers of the encoder neural network of the marked-curve reconstruction model.

layer	output shape	# params	size
linear	$(B, L/2)$	8256	$L/2$
batch norm	$(B, L/2)$	$2(L/2)$	
Rectified Linear Unit (ReLU)	$(B, 2P)$		
linear	$(B, 2P)$	520	$2P$
sigmoid	$(B, 2P)$	520	

Table 2: Summary of the layers of the encoder neural network of the marked-curve reconstruction model.

3.2.3 Marked-Curve Reconstruction Model Invocation

In order to vectorize the entire line-art image, the marked-curve reconstruction model has to be invoked iteratively until all curves are reconstructed. This is done in multiple steps, which are laid out in Algorithm 1.

Note, that Line 5 in Algorithm 1 constitutes an intuitive stopping criterion enabled by the progressive canvas image subtraction from the remaining image. Since missing a few curves is not a significant issue and errors in the model output are to be expected, the stopping criterion is set to $T = \lfloor B * 0.1 \rceil$, where B is the number of black pixels in the original image.

4 Dataset

The dataset used in this work consists of two parts: a human-generated dataset of 20,564 clean line-art images and a synthetic dataset. The human-generated dataset consists of 139 vector images provided by Tonari Animation, 425 vector images from the SketchBench benchmark, and 20,000 amateur sketches from the TU Berlin collection. The size of this dataset is increased using four data augmentation techniques: curve mirroring, curve rotation, curve reversion and curve dropout. The synthetic dataset is used to further increase the size of the training data. For that, images with a low number of randomly sampled cubic Bézier curves are generated and combined with the human-generated dataset at a 1:5 ratio. The entire dataset consists of Scalable Vector Graphics (SVG) vector images and corresponding rasterized Portable Network Graphics (PNG) images, with a uniform color for the background (white) and the curves (black).

5 Evaluation

To answer the RQ1, this section provides a quantitative and qualitative evaluation of the extent to which the line-art vectorization method developed in this work and comparable state-of-the-art methods are able to automatically vectorize clean animation frame line art. It is performed on a held-out portion of the dataset consisting of 10 Tonari animation frames.

5.1 Quantitative Evaluation

To perform the quantitative evaluation, the methods are applied to vectorize a test dataset consisting of evaluation dataset images and their results are compared using metrics which quantify the difference between the ground truth (i.e., the gold standard) and the vectorization results.

In detail, the vectorization methods are given a raster image $\mathbf{X}_{\text{raster}}$ as input and produce an output vector image $\hat{\mathbf{Y}}$, where $\hat{\mathbf{Y}} = (\hat{\mathbf{y}}_j)_{j=0}^{n}$ is a sequence of cubic Bézier curves of arbitrary length n and each cubic Bézier curve $\hat{\mathbf{y}} = (\hat{y}_i)_{i=1}^{8}$ is a sequence of 8 numbers, which represent the curve parameters (i.e., the start point, end point and two control points). The metrics measure how well $\hat{\mathbf{Y}}$ matches the ground truth vector image $\mathbf{Y}$ corresponding to the input image $\mathbf{X}_{\text{raster}}$, where again $\mathbf{Y} = (\mathbf{y}_j)_{j=0}^{m}$ is a sequence of cubic Bézier curves of length m.

Intersection-over-Union (IoU) Following related works [3, 10, 5], the visual similarity of the output image to the ground truth image is measured using the IoU defined in Equation (2). Note, that TP refers to the true positives, FP to the false positives and FN to the false negatives, which are calculated by binarizing the rasterized output image $\hat{\mathbf{Y}}_{\text{raster}}$ and input image $\mathbf{X}_{\text{raster}}$.

$$J = \frac{TP}{TP + FP + FN} \tag{2}$$

Curve error One method of measuring the correctness of the vector structure of the output $\hat{\mathbf{Y}}$ is to calculate its distance to the ground truth image $\mathbf{Y}$. This *curve error* is defined in Equation (3) as the sum of the distance of each curve in the output image to the corresponding ground truth curve. Hungarian ordering is used to establish curve correspondence, i.e. the ground truth curve with the minimum distance is paired to each output curve. The sum of absolute errors defined in Equation (4) is used as distance function $d(\hat{\mathbf{y}}_i, \mathbf{y}_j)$.

$$\text{curve error}(\hat{\mathbf{Y}}, \mathbf{Y}) = \text{mean}_{i=0}^{n} \left(\min_{j=0}^{m} d(\hat{\mathbf{y}}_i, \mathbf{y}_j) \right) \tag{3}$$

$$d(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=0}^{8} |\hat{y}_i - y_i| \tag{4}$$

Curve ratio Given an output image $\hat{\mathbf{Y}}$ that visually resembles the ground truth $\mathbf{Y}$, a simple measure of matching vector structures is to consider the ratio number of output curves and ground truth curves $n/m \in [0, n]$. In the case of perfectly matching vector structures, $n = m$ and $n/m = 1$.

Curve length Another method to measure how well the output vector structure matches the ground truth is to calculate the average curve length in pixels and compare it to the ground truth.

Curve distance Following Yan et al. [20], holes between curves are measured using the the minimum distance of each curve endpoints to each other curve endpoints. In detail, the metric is defined by Equation (5), where $\mathscr{E} = [0, 1, 6, 7]$ defines the indices of the start and the end point parameters of a curve. The closer the value is to the ground-truth baseline, the closer the vector structure can be considered to match the ground truth, while values that are higher than the baseline indicate more unintentional holes.

$$\text{mean}_{i=0}^{n} \left(\min_{j=0}^{n} \sum_{k \in \mathscr{E}} |\hat{y}_k^i - \hat{y}_k^j| \right) \tag{5}$$

Efficiency Furthermore, the runtime (in seconds) and GPU memory usage is measured to evaluate the efficiency of the algorithms.

5.1.1 Results

Table 4 shows the performance of the following line-art image vectorization methods: the method developed in this work (`marked`), the traditional algorithm by Weber [19] (`autotrace`), the vectorization algorithm combining deep learning and heuristic optimization by Puhachov et al. [12] (`polyvector-flow`), the deep learning-based algorithm using raster supervision by Mo et al. [10], (`virtual-sketching`), and the deep learning-based algorithm using vector supervision by Egiazarian et al. [3] (`deepvectechdraw`).

The methods are applied on the Tonari clean animation frame test dataset rasterized at a resolution of 512px, while preserving the aspect ratio. The Intersection-over-Union (IoU), curve error and runtime metrics can be easily interpreted: While the arrow in the column name indicates whether larger or smaller numbers represent better performance, the results of the best and the second-best performing method on the metric are indicated using bold and italics fonts, respectively.

For the remaining metrics, recall that the average curve length and the average curve distance should be close to the ground truth values, which are listed in Table 3. The curve ratio is calculated with the number of curves listed in the same table.

Table 4 shows that the line-art vectorization method developed in this work outputs vector images that resemble the input raster image the closest. It achieves this with the second-smallest curve error behind the method by Puhachov et al. [12] and with a curve distance that is close to the ground truth, just behind the method by Mo et al. [10]. Interestingly, it uses roughly half the curves of the ground truth, with curves on average being nearly twice as long. Finally, it is also the fastest deep learning-based method, while requiring the least amount of dedicated GPU memory.

Note that the traditional method by Weber [19] significantly outperforms all other methods on the runtime. On the other hand, it has the highest curve error and lowest IoU, suggesting ill-fitting outputs. The method by Puhachov et al. [12] also achieves a surprisingly low IoU, but also the best curve error.

The two deep learning-based methods by Mo et al. [10], Egiazarian et al. [3] approach the IoU of the method developed in this work, albeit with a significantly higher curve error and runtime. Additionally, the method by Egiazarian et al. [3] outputs the lowest amounts of curves, but the curves of the method by Mo et al. [10] are still longer on average, suggesting that this method produces more curves that do not fit the ground truth curves.

In general, most methods produce output images that surprisingly do not cover the input image well. This suggests that no method reproduces clean animation frames to the extent required by the task considered in this work.

5.1.2 Results with higher resolution input images

The methods by Weber [19], Puhachov et al. [12] performed unusually low on the evaluation in Table 4. A potential cause for this was identified as the low resolution of input images at 512px. To investigate this hypothesis, the evaluation was rerun with input images rasterized at twice the resolution, i.e., 1024px, while preserving the aspect ratio. Keep in mind that this is significantly higher than the standard resolution of clean animation frames considered in this work. Hence, performance increases of methods at this resolution will likely not materialize when they are applied to real-world clean animation frames, which usually will only be available at a lower resolution.

Figure 4 compares the evaluation results of the two resolutions sizes. Note that, since metrics measured in pixels

		curves median	curve length median	curve distance median
tonari	512-0.512	205.00	2.56	1555.82
	1024-1.024	205.00	5.12	1725.81
sketchbench	512-0.512	208.00	12.43	2355.22
	1024-1.024	208.00	24.85	2726.03

Table 3: Selected metrics of the vector images in the test dataset. This information can be used as baseline for the corresponding metrics in Table 4.

		autotrace	polyvector-flow	virtual-sketching	deepvec-techdraw	marked (ours)
IoU $\uparrow$	median	0.02	0.12	*0.29*	0.28	**0.30**
curve ratio	median	0.23	1.35	0.30	0.19	0.43
curve length	median	1.00	0.55	11.16	9.06	8.19
curve distance	median	891.00	439.18	1442.91	917.50	1361.28
curve error $\downarrow$	median	20.37	**14.05**	20.08	17.58	*16.76*
runtime $\downarrow$	median	**0.35**	14.82	22.99	97.73	*9.49*

Table 4: Comparison of the performance of the marked line-art image vectorization method and four prior works on the Tonari test subset at a resolution of 512px. If possible, the result of the best and the second-best performing method for the metric is indicated using bold and italics fonts, respectively.

scale linearly with the resolution size, they are normalized by the resolution size. It is clear that all prior methods except AutoTrace [19] perform significantly better than at 512px resolution. The method by Puhachov et al. [12] even reaches an IoU well over 0.5, i.e., its outputs cover more than half of the input image correctly on average. This is dampened by a high curve error and curve distance, indicating incorrect vector structures. The method by Egiazarian et al. [3] performs similarly well, with a lower IoU but better curve error and curve distance, seemingly striking a different balance between visual resemblance and semantically correct vector structures.

Interestingly, the metrics of the method developed in this work stay remarkably stable at the increased resolution. This is especially remarkable for the runtime, which significantly and predictably changes for all other methods.

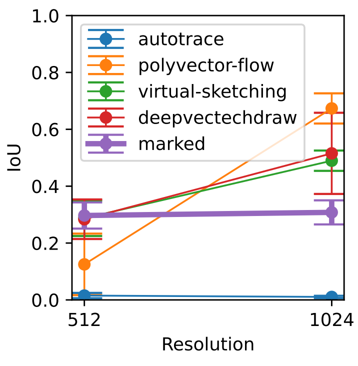
One potential reason for this remarkable input image resolution invariance of the method developed in this work is the selection of reconstruction curves using marks, which explicitly *forces* the model to reconstruct curves which other methods might not have detected. This can be the case for curves that are too thin or contain some spots at low resolutions.

5.2 Qualitative Evaluation

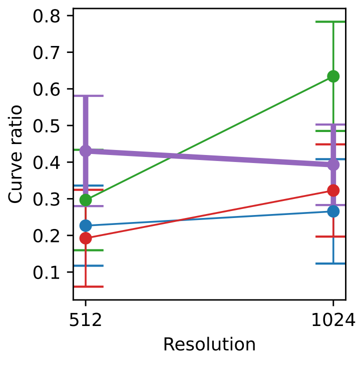
For a visual comparison, Figure 5 shows the best output of each method for an example clean animation frame by Tonari Animation. The input image has a resolution of 512px and is binarized for the methods by Weber [19], Puhachov et al. [12], Mo et al. [10], since that leads to higher-quality outputs. Since the main objective is to not only achieve visual similarity but also match the semantically correct vector structure of the ground truth vector image, Figure 5 attempts to visualize the underlying vector structure. Following Guo et al. [5], Mo et al. [10], Puhachov et al. [12], this visualization is achieved by representing curves using mutually exclusive colors. Furthermore, the images are zoomed in to lay bare minute differences. Indications for a correct vector structure are a constant color for continuous curves and a similarity to Figure 5a.

All methods appear to be visually correct at first glance, with varying quality and the methods by Egiazarian et al. [3], Mo et al. [10] not performing favourably. However, looking into details reveals significant deficiencies. The method developed in this work arguably produces the most closely matching vector structure, with most curves faithfully reconstructed following their appearance. On the other hand, curves are often slightly too short, leaving undesirable holes. Furthermore, there is a bias towards low curvature.

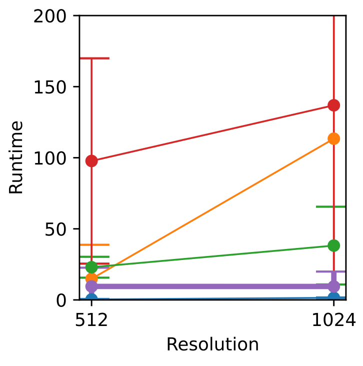
The method by Mo et al. [10] is similar to the method developed in this work in that it faithfully reconstructs curves, but fails to preserve the constant stroke width. The methods by Weber [19], Puhachov et al. [12] do not faithfully reconstruct curves, with multiple curves often merged into a single curve or altogether missing. This leads to a visually clean output – even without a significant amount of holes in the case of AutoTrace [19]. However, the produced vector structure is far from the ground truth in Figure 5a.



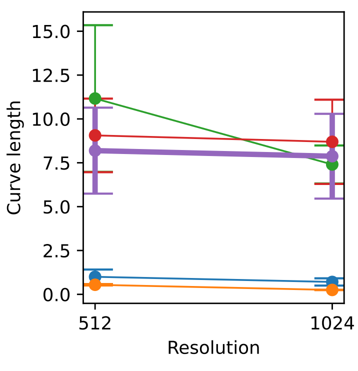
(a) The average IoU.



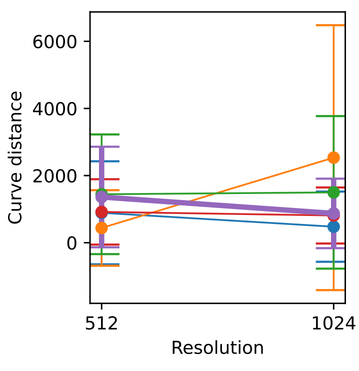
(b) The average curve ratio.



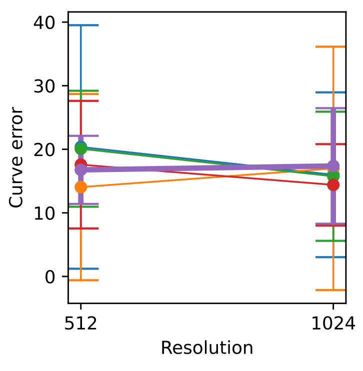
(c) The average runtime.



(d) The average curve length.



(e) The total curve distance.



(f) The average curve error.

Figure 4: Metrics for the line-art image vectorization methods evaluated on images with 512px and 1024px resolution, respectively. Points denote the median of the metric, while vertical bars denote the inter-quartile range (IQR). Horizontal lines show the trend of the metric. The metrics for the method developed in this work are emphasized. Note that they are not significantly affected by the image resolution and none decreases with lower resolutions.

6 Conclusion

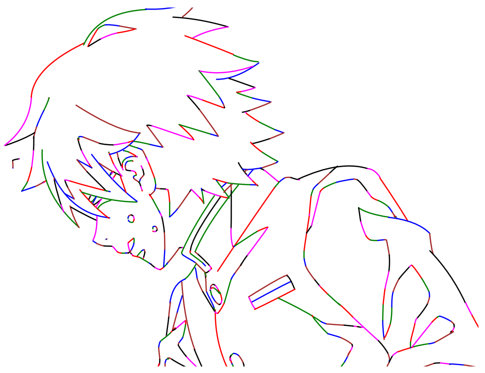
The objective of this work was to ascertain to what extent it is possible to automatically vectorize clean animation frame line art in a semantically meaningful way. In order to answer the RQ1, Section 3 proposed a clean animation frame line-art image vectorization method and Section 5 evaluated it together with prior work on an evaluation dataset provided by Tonari Animation. It could be shown that while the proposed method outperforms prior work at the default input image resolution, ultimately no line-art image vectorization method is able to satisfactorily vectorize clean animation frames, especially failing to properly reconstruct details and primitives with high curvature. Hence, no method studied in this work is of practical use in the limited-animation workflow. In order to achieve the goal of automatizing the tedious step of vectorizing clean animation frames, the curve reconstruction needs to be significantly more accurate.

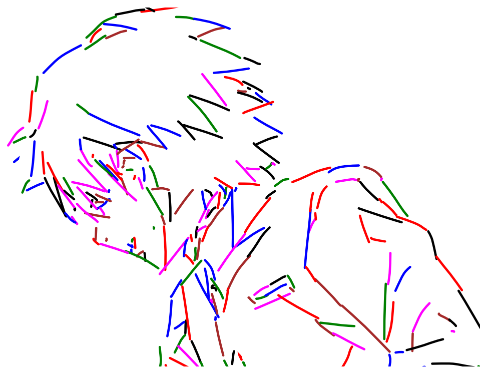
Advantages of the developed method include remarkable robustness to input image resolution and binarization, resource efficiency and flexibility for manual fixing. Limitations include a significant amount of small holes in reconstructed curve sequences, limited semantic correctness and a bias towards lower curvature.

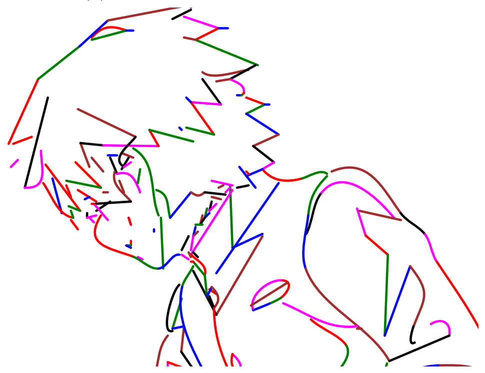
6.1 Future Work

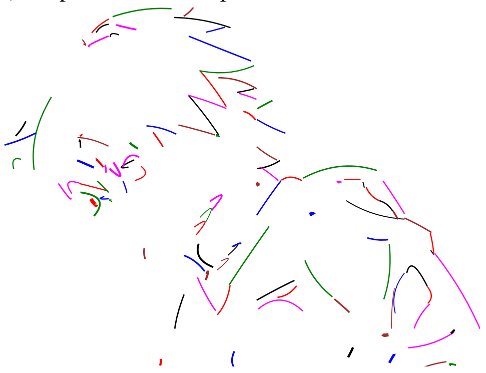
There are numerous opportunities to improve on the presented work. The dataset could be improved by collecting a larger amount of high-quality data or performing more advanced data augmentation or feature extraction. A further promising improvement is to finetune a large vision-language model such as CogVLM [17] instead of training a small CNN based encoder-decoder model from scratch in order to utilize their emergent capabilities.

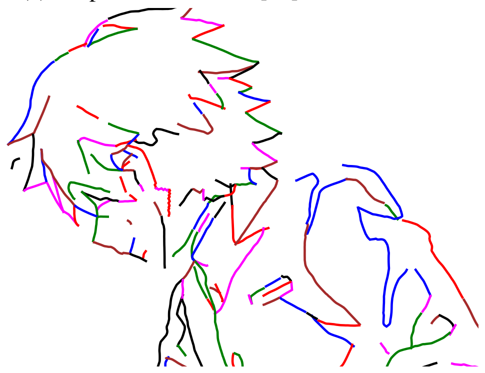
Furthermore, there exist other tasks to which the developed model could be extended. These include the generation of inbetween frames based on keyframes or clean animation frame colorization. Moreover, the model output could be constrained to exhibit temporal consistency, i.e., to consist of curves that remain consistent across frames of the same scene.


(a) Ground truth.


(b) Output of the developed method.


(c) Output of AutoTrace [19].


(d) Output of Egiazarian et al. [3].


(e) Output of Puhachov et al. [12].

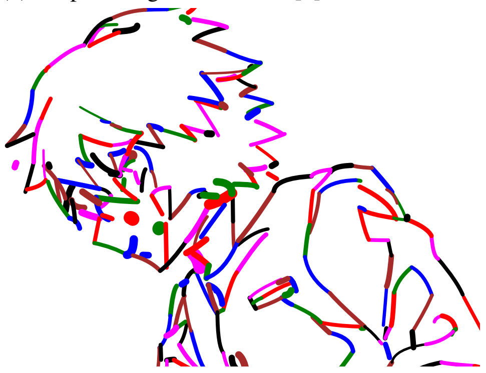

(f) Output of Mo et al. [10].

Figure 5: The output vector image given a Tonari clean animation frame in raster format as input of each line-art image vectorization method studied in this work. The vector structure behind the images is revealed by representing each curve with a mutually exclusive color and a high zoom level.

References

[1] Mikhail Bessmeltsev and Justin Solomon. Vectorization of line drawings via polyvector fields. *ACM Trans. Graph.*, 38(1):9:1–9:12, 2019. doi: 10.1145/3202661. URL https://doi.org/10.1145/3202661.

[2] Ayan Kumar Bhunia, Pinaki Nath Chowdhury, Yongxin Yang, Timothy M. Hospedales, Tao Xiang, and Yi-Zhe Song. Vectorization and rasterization: Self-supervised learning for sketch and handwriting. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 5672–5681. Computer Vision Foundation / IEEE, 2021. doi: 10.1109/CVPR46437.2021.00562. URL https://openaccess.thecvf.com/content/CVPR2021/html/Bhunia_Vectorization_and_Rasterization_Self-Supervised_Learning_for_Sketch_and_Handwriting_CVPR_2021_paper.html.

[3] Vage Egiazarian, Oleg Voynov, Alexey Artemov, Denis Volkhonskiy, Aleksandr Safin, Maria Taktasheva, Denis Zorin, and Evgeny Burnaev. Deep vectorization of technical drawings. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XIII*, volume 12358 of *Lecture Notes in Computer Science*, pages 582–598. Springer, 2020. doi: 10.1007/978-3-030-58601-0_35. URL https://doi.org/10.1007/978-3-030-58601-0_35.

[4] Jun Gao, Chengcheng Tang, Vignesh Ganapathi-Subramanian, Jiahui Huang, Hao Su, and Leonidas J. Guibas. Deepspline: Data-driven reconstruction of parametric curves and surfaces. *CoRR*, abs/1901.03781, 2019. URL http://arxiv.org/abs/1901.03781.

[5] Yi Guo, Zhuming Zhang, Chu Han, Wenbo Hu, Chengze Li, and Tien-Tsin Wong. Deep line drawing vectorization via line subdivision and topology reconstruction. *Comput. Graph. Forum*, 38(7):81–90, 2019. doi: 10.1111/cgf.13818. URL https://doi.org/10.1111/cgf.13818.

[6] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II*, volume 9906 of *Lecture Notes in Computer Science*, pages 694–711. Springer, 2016. doi: 10.1007/978-3-319-46475-6_43. URL https://doi.org/10.1007/978-3-319-46475-6_43.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[8] Tzu-Mao Li, Michal Lukác, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph.*, 39(6):193:1–193:15, 2020. doi: 10.1145/3414685.3417871. URL https://doi.org/10.1145/3414685.3417871.

[9] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.4400.

[10] Haoran Mo, Edgar Simo-Serra, Chengying Gao, Changqing Zou, and Ruomei Wang. General virtual sketching framework for vector line art. *ACM Trans. Graph.*, 40(4):51:1–51:14, 2021. doi: 10.1145/3450626.3459833. URL https://doi.org/10.1145/3450626.3459833.

[11] Gioacchino Noris, Alexander Hornung, Robert W. Sumner, Maryann Simmons, and Markus H. Gross. Topology-driven vectorization of clean line drawings. *ACM Trans. Graph.*, 32(1):4:1–4:11, 2013. doi: 10.1145/2421636.2421640. URL https://doi.org/10.1145/2421636.2421640.

[12] Ivan Puhachov, William Neveu, Edward Chien, and Mikhail Bessmeltsev. Keypoint-driven line drawing vectorization via polyvector flow. *ACM Trans. Graph.*, 40(6):266:1–266:17, 2021. doi: 10.1145/3478513.3480529. URL https://doi.org/10.1145/3478513.3480529.

[13] Pradyumna Reddy. Im2vec: Synthesizing vector graphics without vector supervision. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2021, virtual, June 19-25, 2021*, pages 2124–2133. Computer Vision Foundation / IEEE, 2021. doi: 10.1109/CVPRW53098.2021.00241. URL https://openaccess.thecvf.com/content/CVPR2021W/SketchDL/html/Reddy_Im2Vec_Synthesizing_Vector_Graphics_Without_Vector_Supervision_CVPRW_2021_paper.html.

[14] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992. doi: 10.1162/NECO.1992.4.1.131. URL https://doi.org/10.1162/neco.1992.4.1.131.

[15] Peter Selinger. Potrace: a polygon-based tracing algorithm, September 2003. URL https://www.mathstat.dal.ca/~selinger/potrace/potrace.pdf.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[17] Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. Cogvlm: Visual expert for pretrained language models. *CoRR*, abs/2311.03079, 2023. doi: 10.48550/ARXIV.2311.03079. URL https://doi.org/10.48550/arXiv.2311.03079.

[18] Yizhi Wang and Zhouhui Lian. Deepvecfont: synthesizing high-quality vector fonts via dual-modality learning. *ACM Trans. Graph.*, 40(6):265:1–265:15, 2021. doi: 10.1145/3478513.3480488. URL https://doi.org/10.1145/3478513.3480488.

[19] Martin Weber. AutoTrace, September 2002. URL https://autotrace.sourceforge.net/. accessed on 2022-12-03.

[20] Chuan Yan, David Vanderhaeghe, and Yotam Gingold. A benchmark for rough sketch cleanup. *ACM Transactions on Graphics (TOG)*, 39(6), November 2020. ISSN 0730-0301. doi: 10.1145/3414685.3417784. URL https://doi.org/10.1145/3414685.3417784.

[21] Zibo Zhang, Xueting Liu, Chengze Li, Huisi Wu, and Zhenkun Wen. Vectorizing line drawings of arbitrary thickness via boundary-based topology reconstruction. *Computer Graphics Forum*, 41(2):433–445, 2022. doi: https://doi.org/10.1111/cgf.14485. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14485.

A Appendix

This section contains additional information related to the implementation of the vectorization algorithm.

Algorithm 1: Iterative Curve Reconstruction.

```
   Input: A raster line-art image.
   Output: A vector line-art image.
1  Segment input image by color;
2  foreach image segment do
3      canvas = an empty vector image of the same size
         as the input image;
4      remaining = image segment;
5      while number of black pixels in remaining $> T$;
         do
6          Compute marker by applying curve
             identification on the remaining image;
7          Centered image = center the remaining
             image on the marker;
8          reconstructed curve = invoke the
             marked-curve reconstruction model using
             the centered image;
9          Inverse the center location of the curve by
             using the mark location;
10         Add the reconstructed curve to the canvas
             image;
11         remaining = remaining - rasterized canvas
             image;
12     end
13     Set color of all curves in the canvas image to the
         segment color;
14 end
15 Merge the canvas images;
16 return Merged canvas images
```