TU WIEN Informatics

Semantically Meaningful Vectorization of Line Art in Drawn Animation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Calvin Metzger
Matrikelnummer 1454634

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Projektass. Joao Afonso Cardoso, MSc

SIGNATURE INFORMATION	Signatory	Michael Johann Laurenz Wimmer
	Date/Time-UTC	2023-12-03T23:16:35+01:00
	Verification	Information about the verification of the electronic signature can be found at: https://www.signaturpruefung.gv.at
Note	This document is signed with a qualified electronic signature. According to Art. 25 para. 2 of the Regulation (EU) No 910/2014 of 23. July 2014 ("eIDAS-Regulation") it shall have the equivalent legal effect of a handwritten signature.	

Wien, 3. Dezember 2023

Calvin Metzger Michael Wimmer

Semantically Meaningful Vectorization of Line Art in Drawn Animation

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Calvin Metzger
Registration Number 1454634

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Projektass. Joao Afonso Cardoso, MSc

Vienna, 3rd December, 2023

Calvin Metzger   Michael Wimmer

Erklärung zur Verfassung der Arbeit

Calvin Metzger

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. Dezember 2023

Calvin Metzger

Acknowledgements

Thanks to Joao Afonso Cardoso for his help in finding research ideas, setting up a connection to Tonari Animation and providing continuous guidance and assistance for both the thesis and the underlying research.

Thanks to Michael Wimmer for supervising the thesis and providing feedback.

Thanks to Jarrett Martin and Tonari Animation for providing a valuable dataset of clean animation frames for this research.

Thanks to Yukinobu Taniguchi for providing ideas and feedback on the research and general support during my stay abroad at the Tokyo University of Science.

Thanks to Siegfried Höfinger, the TU.it dataLab team and the System Administration team of the Computer Graphics Research Unit for providing the computing resources required for this research.

Thanks to my parents, family and friends for their support throughout this time.

Kurzfassung

Animation besteht aus der sequenziellen Darstellung mehrerer Einzelbilder mit geringfügigen mutualen Unterschieden um den visuellen Effekt einer Bewegtbildszene zu erreichen. In der sogenannten limitierten Animation werden diese Einzelbilder als Vektorgraphiken mit spezifischer semantischer Bedeutung gezeichnet. Diese können als reine Animationsbilder bezeichnet werden. In manchen Arbeitsabläufen sind diese reinen Animationsbilder lediglich als Rastergraphiken vorhanden, was eine aufwändige manuelle Vektorisierung dieser erfordert.

Diese Arbeit erkundet in welchem Umfang Methoden zur Vektorisierung von Liniengraphiken verwendet werden können, um diesen Prozess zu automatisieren. Zu diesem Zwecke wird eine Liniengraphikvektorisierungsmethode entworfen und entwickelt, welche die strukturellen Informationen reiner Animationsbilder berücksichtigt. Zusammen mit existierenden, modernen Methoden wird diese Methode anhand eines Datensatzes von reinen Animationsbildern evaluiert. Diese reproduzierbare Evaluierung zeigt, dass die Performanz der entworfenen Methode über verschiedene Auflösungen von Eingabebildern und binären und nicht-binären Versionen von Eingabebildern hinweg erstaunlich stabil bleibt. Zudem liefert die Methode bessere Resultate als bestehende moderne Methoden für Eingabebilder mit der standardmäßigen Auflösung von reinen Animationsbildern. Weiters ist sie bis zu 4.5-Mal so schnell als die zweitschnellste Deep Learning-basierte Methode. Nichtsdestotrotz zeigt die Evaluierung, dass weder die entwickelte Methode noch bestehende Methoden Vektorgraphiken mit ausreichend visueller Ähnlichkeit und semantisch korrekten Vektorstrukturen produzieren kann.

Abstract

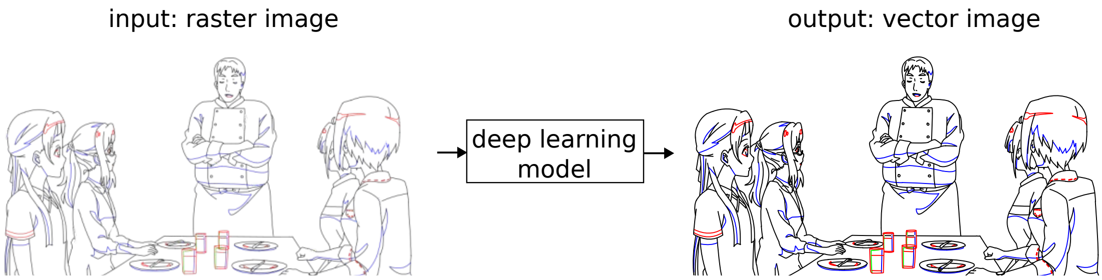
Animation consists of sequentially showing multiple single frames with small mutual differences in order to achieve the visual effect of a moving scene. In limited animation, these frames are drawn as semantically meaningful vector images which could be referred to as clean animation frames. There are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

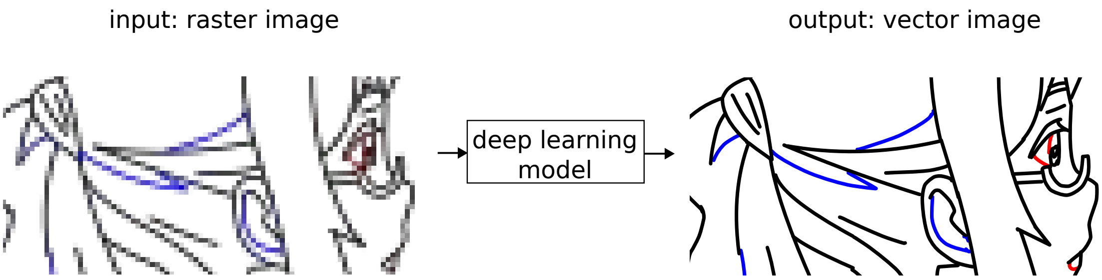
Contents

CHAPTER 1

Introduction



(a) The general design of the proposed objective. Input and output images are provided by Tonari Animation.



(b) Highlighted section of input and output images, which reveals structural differences.

Figure 1.1: 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. 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 4.1

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. While both full animation and limited animation techniques reuse some fixed parts (like the background), the reusing of cels by limited animation leads to the characteristic stiff style and reduces the cost to produce an animated series. Today, the technique is primarily used in hand-drawn animations, often under the term *anime* popularized by Japanese media.

While limited animation was mostly replaced with full animation and 3D animation in the western world, it still enjoys continued relevance due to the growing importance of anime. This importance is underlined by Napier [2016], who estimates that anime made up over 60 percent of all animated shows in 2016. Unsurprisingly, anime is especially prevalent in Japan, where it accounts for 6 of the top 10 highest-grossing films in 2021 [Motion Picture Producers Association of Japan, Inc, 2021]. According to Masuda et al. [2023], the worldwide market for anime monotonically grew from 1266.1 billion Yen in 2009 to 2742.5 billion Yen in 2021 (with an exception in the year 2020). This trend suggests that limited animation will continue to be relevant in the future.

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 keyframes are vectorized and cleaned. To achieve the visual effect of fluidity, a large number of frames in between the keyframes are drawn. In contrast to full animation, these *inbetweens* are not redrawn completely, but reuse the majority of a keyframe while only editing small parts of the image. Finally, in the fourth phase, the clean frames are colored and enriched with special effects and a background image.

Thus, the production consists of creative parts such as the drawing of keyframes and tedious tasks such as the creation of inbetweens. The latter tasks are a ripe target for automatization, freeing up animation studios to focus on the creative parts of their work. This work explores the automatization of one such task, specifically the vectorization of clean animation frames.

Traditionally, computational tasks are automated using hand-crafted algorithms with manually set parameters. While this is feasible for well-defined and structured tasks, it is unfeasible for complex, ambiguous tasks with high-dimensional data, such as the vectorization of clean animation frames. A different automatization technique that is better fit for these types of tasks is *machine learning*. Contrary to hand-crafted algorithms, machine learning consists of assembling a large dataset of inputs and outputs and defining an algorithm which exploits statistical correlations in this dataset to fit a model that generalizes to data outside of this dataset. A subset of machine learning that has increased in importance in recent years is *deep learning*, in which the model fit to the dataset is an artificial neural network [Rosenblatt, 1958].

In this chapter, Section 1.1 provides the motivation for the research. Section 1.2 details the research question and related research objectives. Section 1.3 lists the challenges associated with the research objectives. Section 1.4 gives a summary of the main contributions of this work. Finally, Section 1.5 outlines the structure of the thesis.

1.1 Motivation

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. Contrary to raster images, which represent an image using a raster of color values, vector images represent an image using a hierarchy of graphical primitives. The primitives used in this case are lines and curves. Having the clean frame as vector line art enables accurate and easy editing. Furthermore, vector files are resolution-independent and require less data to represent an image than lossless raster formats, where the color of each pixel needs to be stored somehow. Moreover, by their nature they also contain the semantic composition of the image, which is potentially useful for downstream tasks.

In many animation studios, limited animation frames are initially drawn on paper, i.e., in raster format. Afterwards, these frames need to be manually traced in order to produce clean frames in vector format. This manual conversion is a tedious and time-intensive process, requiring roughly an hour per frame. Hence, it would be beneficial to have a method which can convert the clean frame into vector format automatically.

Even disregarding the step from rough animation frames in raster format to clean frames in vector format, there are scenarios where the clean frame is only available as a raster image. This could be due to various reasons, including discontinuity in the production workflow (e.g., when different studios are contracted for different steps), the need to export an image from the drawing program to another application, or the existence of old production data that needs to be reused.

Once the clean animation frame is available in vector format, it increases the efficiency of successive workflow steps. An example is the creation of the frames inbetween keyframes, where artists can naturally utilize existing keyframes and only edit a small part of the image. As an example, the curves making up strains of hair could simply be moved instead of needing to be completely redrawn in the case of a raster image.

In summary, 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. Therefore, this work explores the possibility of automatizing this laborious process using line-art image vectorization methods.

1.2 Aim of the Work

The purpose of this work is to create a method to automatically convert clean animation frame raster images into corresponding vector images. As mentioned in Figure 1.1, this method consists of a deep learning model trained on clean animation frames. For the resulting line-art vector image to be useful for artistic purposes, the content needs to be semantically meaningful, i.e., the arrangement, topology and parameterization of graphical primitives (i.e., lines and curves) need to make sense and be close to how artists would draw. This prevents traditional algorithms [Selinger, 2003, Weber, 2002, Noris et al., 2013] to be used for that purpose, as these often produce vector images that visually resemble the raster image closely, but contain semantically meaningless vector primitives - for example, a naive but visually convincing vectorization solution is to represent each pixel as a small line.

In detail, this work will attempt to answer a

Research Question 1 (RQ1) , which is to what extent is it possible to automatically vectorize clean animation frame line art in a manner that is semantically meaningful?

1.3 Challenges

Creating the deep learning-based method to vectorize clean animation frames represents a challenge in itself, due to the visual structure of the images. Specifically, it is crucial that the output of the proposed deep learning model is structurally similar to real clean frame vector images, i.e., that the output consists solely of the primitives that artists use to draw clean animation frames. Hence, it is necessary to find a representation of these primitives that is suitable for deep learning models.

Another challenge is the low amount of available data. While a small dataset ($N = 139$) of raster/vector image pairs of clean frames was provided to us as part of a research project (and drawn at the Tonari Animation studio), this is not a sufficient quantity to fit a deep learning model. Since there is little publicly available data for this task, extending the dataset is non-trivial. In order to achieve this, data augmentation methods have to be used in addition to procuring public data.

Moreover, clean animation frame images often contain a large amount of curves (usually more than 1000 lines per image). The proposed solution needs to be designed in a way to handle this large amount of primitives, since prior deep learning-based works are often suited only for images with a lower amount of curves. Additionally, it is necessary for the image vectorization method to be applicable at lower image resolutions, since clean frame raster images (especially older production data) are often at a resolution for which existing algorithms were not designed. Furthermore, it is unavoidable that the resulting vector images will contain errors and therefore need to be corrected manually. Hence, the solution should focus more on getting the curves it generates correct, rather than covering all curves in the input image. In other words, a solution that generates half of the curves completely correct is preferable to a solution that generates all of the curves only somewhat correct, since in the latter case all curves have to be manually corrected anyway.

Finally, the clean animation frame vectorization method should be designed as a deep learning model, as this makes it easier to adapt it to vectorize raster images from different domains using finetuning. This is important for a related task, which is motivated in the following: One factor that has greatly limited successful attempts in the field of deep learning in hand-drawn limited animation production is the scarce amount of available production data that can be used for training. While it is probably fruitless to attempt to design a system that generates the whole animation automatically, solving individual steps in the production process might be possible. It is clear that training such models would require a large amount of production data, specifically production data that is in reverse order of the actual production workflow. As an example, if a large dataset of clean keyframes and associated inbetweens (as well as animation timesheets) was available, it would be possible to train an inbetween generation model. Other potential uses include clean frame coloring, inpainting or compositing.

While the obvious way of attaining an animation production dataset would be for a studio to publish it, thus far no such dataset has been published, which presents another challenge. An alternative would be to synthesize the data. The differentiable nature of the proposed line-art vectorization algorithm could be utilized for that purpose. While the model will be trained using clean frames as input data, it might be possible to extend or finetune the model to use images of another domain (such as final animation frames) as input. Then, it could be used for a cross-domain vectorization model in order to convert raster images of one domain (e.g., final animation frames) into vector images of another domain (i.e., clean frames). Figure 1.2 depicts such a model. Since final animation frames are readily available in high quantity and quality, this would potentially make it possible to synthesize a large dataset of clean frames requiring only small supervision. However, creating such a model is not the focus of this work, rather, the above-mentioned line-art vectorization method is designed in a way such that it could in principle be extended to cross-domain line-art vectorization. Most importantly, this means that the model is nearly end-to-end differentiable, such that it can be easily finetuned on other data. This is a characteristic that traditional algorithms [Selinger, 2003, Weber, 2002, Noris et al., 2013] do not possess.

In summary, the quantity of the data and the qualitative structure of the data domain impose a range of restrictions which need to be accounted for in the architecture design of the method. Furthermore, the method should be based on a deep learning model in order to enable adaptability to other input data domains.

1.4 Contributions

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 described in Section 4.1 and based on a deep learning model to enable it to be adapted to different input image domains. Furthermore, it is designed to fit the qualitative structure of clean animation frames as input and output images.

Accordingly, the Research Objective 2 (RO2) is to perform an evaluation that ascertains

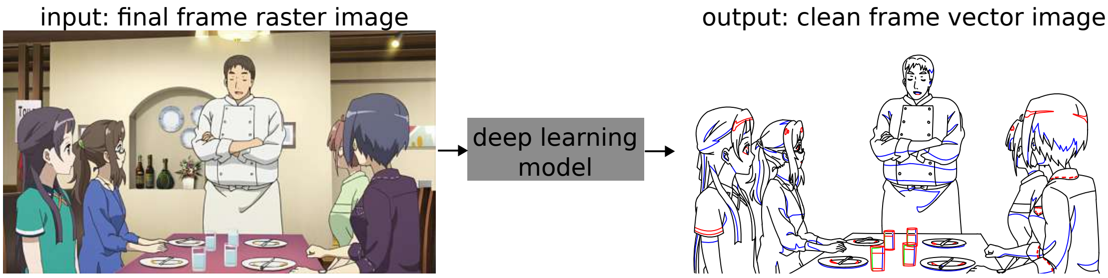


Figure 1.2: An overview of cross-domain line-art vectorization, a potential extension of the proposed solution. Here, the objective is to convert final animation frame raster images (example by Celibidache and P.A. WORKS [2017]) into clean animation frame vector images by extending or finetuning the deep learning model depicted in Figure 1.1.

the extent to which the developed method and existing state-of-the-art line-art image vectorization methods are able to vectorize clean animation frames. This evaluation is described in Section 4.3. Great care is taken to ensure that this evaluation is reproducible. Due to this, the whole evaluation is not only conducted on the clean animation frames provided by Tonari Animation, but also on a publicly available subset of the Sketch-Bench [Yan et al., 2020] dataset. Furthermore, the code for the evaluation is publicly available at `https://github.com/nopperl/marked-lineart-vectorization`. Ultimately, while the evaluation shows advantages of the developed line-art image vectorization method compared to existing state-of-the-art methods, no method is able to satisfactorily vectorize clean animation frames.

RO1 A deep learning-based method for clean animation line-art vectorization.

RO2 A reproducible evaluation assessing the extent to which line-art image vectorization methods can vectorize clean animation frames.

1.5 Structure

Chapter 2 describes the necessary prerequisites required to understand the proposed solution. Chapter 3 gives an overview of prior work related to the research objectives. The proposed solution is described and empirically evaluated together with state-of-the-art methods in Chapter 4. Chapter 5 revisits the research objectives, lists limitations and provides potential future work.

CHAPTER 2

Background

This chapter details the theoretical background of clean animation frame vectorization. Section 2.1 gives an overview of both raster and vector images as well as the process of vectorization, i.e., converting raster images to vector images. Section 2.2 describes the process for hand-drawn limited animation in more detail. Section 2.3 explains deep learning, which is the technique used in this work to develop an automatic clean animation frame vectorization method.

2.1 Vectorization

In a digital context, images can be represented using either *raster* or *vector* data formats. These are introduced in Sections 2.1.1 and 2.1.2, respectively. This work is concerned with *image vectorization*, i.e., the conversion of raster images into corresponding vector images, which is described in Section 2.1.3.

2.1.1 Raster Images

Digital visual data are most commonly represented using *raster* data formats. Raster images store visual data as a *raster*, a 2-dimensional grid of uniformly sized squares. Each of these squares can be referred to as *pixel* and is defined by its location and its color.

Raster images can be naively stored using numerical two-dimensional arrays or matrices. The pixels location in the array is directly mapped to the location on the grid (and therefore on the display). This way, the only information that needs to be stored explicitly for each pixel is its color. Usually, a color is defined using a *color model* such as the RGB or the CMYK model. In the case of RGB, a color is defined as the intensity of each of the components red, green and blue. These components are then mixed according to their respective intensity using simple addition to produce a color. Using this additive model, a broad spectrum of colors can be represented using only 3 numerical values. This

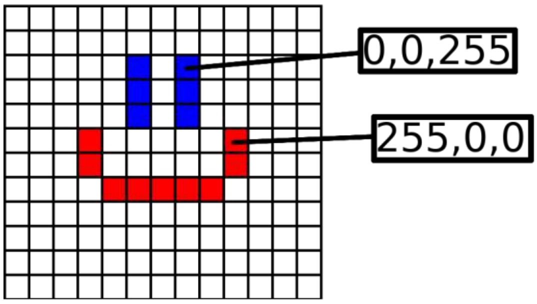


Figure 2.1: Example of an image represented using a raster of 13x12 pixels. Notice that color information is stored explicitly per pixel.

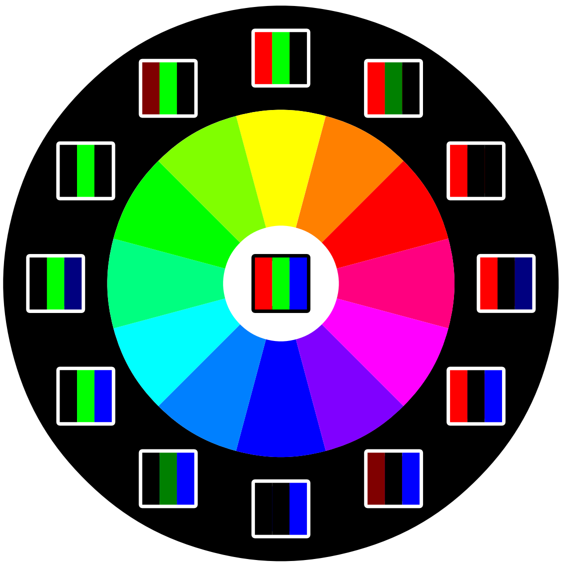


Figure 2.2: A color wheel exemplifying the Red-Green-Blue (RGB) color model. The box next to each color displays the intensities of the red, green and blue component respectively (with black indicating zero intensity). The three components are added in order to produce the displayed color. Created by Németh [2013].

is shown in Figure 2.2. Furthermore, the numerical values are bounded and usually in $[0, 1]$ (representing relative amounts) or in $[0, 255]$ (in absolute amounts, intentionally the maximum amount which can be stored in a bit). This data format is exemplified in Figure 2.1.

There exist multiple ways of storing raster images. The naive storage format mentioned above is referred to as bitmap. Since the location of each pixel of the grid is implicitly given, the only information that needs to be stored is the color (i.e., three numerical values) per pixel. Other formats such as the commonly used Portable Network Graphics (PNG) [Boutell, 1997] or JPEG attempt to reduce the necessary storage space using compression algorithms. The format can also be extended to include opacity (i.e., transparency) as an additional fourth component of the color model (then referred to as RGBA color model). The grid (and in turn the array) can also be extended by a third dimension in order to represent three-dimensional graphics (referred to as *voxel data*).

2.1.2 Vector Images

(a) A vector image consisting of 5 shapes.

(b) A raster image consisting of 50x50 pixels

Figure 2.3: An example image in both vector and raster format. By zooming in, it is possible to experience the fundamental difference between the two formats.

A different way to represent images are vector images. In contrast to raster formats, vector data formats represent images using graphical primitives. These primitives are the most basic geometries which constitute images, such as points, lines or polygons. Using these low-level primitives, it is also possible to define higher-level shapes like text, circles or parametric curves. Each primitive or shape can carry attributes such as the color it is filled with, location or size. These primitives can be arranged both in a hierarchical and in a topological structure, i.e., their spatial relations can be explicitly defined. Figure 2.3 gives an example of an image in both vector and raster format. The vector image in Figure 2.3a is stored using a vector format called Scalable Vector Graphics (SVG) [Bellamy-Royds et al., 2018], which is developed by the World Wide Web Consortium and is widely used, especially on the world wide web. The underlying SVG contents can are shown in Listing 1. Note that, in contrast to storing each pixel and its color, the SVG file stores way less data.

Vector images possess specific advantages over raster images, depending on the context they are used in. The advantages arise from the fact that the image is represented using

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<svg
   width="100"
   height="100"
   version="1.1"
   xmlns="http://www.w3.org/2000/svg"
   xmlns:svg="http://www.w3.org/2000/svg">
<circle
   style="fill:#ffee00;stroke-width:0.264999;fill-opacity:1"
   cx="13.658114"
   cy="13.587115"
   r="10" />
<path
   style="stroke:#000000;stroke-width:0.27px;stroke-linejoin:miter"
   d="m 5.42,14.74 c 3.75,8.27 12.45,8.06 17.34,0.038" />
<ellipse
   style="fill:#0000ff;fill-opacity:1;stroke-width:0.324317"
   cx="11.310536"
   cy="11.553639"
   rx="1.3157237"
   ry="2.5536389" />
<ellipse
   style="fill:#0000ff;fill-opacity:1;stroke-width:0.324317"
   cx="15.776539"
   cy="11.553638"
   rx="1.3157237"
   ry="2.5536389" />
<path
   style="fill:#ff00ff;fill-opacity:1;stroke-width:0.264999"
   d="m 11.728484,16.514472 1.290489,2.182378 -2.535239,0.02641 z" />
</svg>
```

Listing 1: The image displayed in Figure 2.3a as SVG file.

only a few graphical primitives and their mutual relations.

Small storage size A vector image is generally smaller than a raster image, since in contrast to raster images it does not describe each pixel, rather only the primitives and their relations. In general, it holds that if the number of primitives required to represent an image is roughly at least less than the number of pixels, the storage size of the vector image will be lower than the corresponding raster image.

Easy editability The primitives and structure of a vector image can be easily loaded and edited, which is not the case for raster images, which can only be edited at a pixel level.

Resolution-independence Since the vector image only defines graphical primitives without an explicit raster, it is independent of the resolution of the raster it is displayed on. That means that vector images can be zoomed in indefinitely without losing their apparent quality, or smoothness. In contrast, zooming into a raster image will quickly reveal non-smooth parts.

It is important to note that the above-mentioned advantages are not intrinsic to the vector format itself. The nature of vector formats (and in turn, their advantages) can be reduced ad absurdum. For example, consider a vector format which stores images using the graphical primitive of squares only. A size, fill color and location can be defined for each of these squares. If these squares are then arranged in a grid and of uniform size, this vector image is in essence a raster image in vector format. Viewed another way, a raster image could be considered an optimized way to store such a naive vector image, since in raster formats the size and location of each square does not need to be explicitly stored. To conclude this thought experiment, it is important to note that such a naive vector format loses most advantages associated with vector images (i.e., resolution-independence, easy editability and smaller storage size). Hence, the advantages of vector formats are not intrinsic to their representation of images, rather, each specific vector image needs to be defined using primitives that match the semantic composition of the image, not just the visual appearance.

To reiterate the above conclusion, an important characteristic of a good-quality vector image is that it consists of semantically meaningful primitives and structure. That is, a vector image should describe the visual components using appropriate primitives intuitive to humans (in addition to their mutual relations) and not just match the visual appearance perfectly on a pixel-level. Not only does this provide a high-level description of visual content, but it also reduces the storage space required and retains the important resolution-independence. However, it is clear that these advantages are only given if the visual can actually be represented using the graphical primitives given by the vector format. Hence, there is no general superiority of one image format over the other. Rather, whether vector or raster formats are appropriate always depends on the given task.

Figure 2.4: Example of line art. Drawn by Santiago Rial [Yan et al., 2020].

An example of an image domain which is appropriate to be displayed using vector formats is line art. This category subsumes drawings that depict objects using lines and curves contrasted with more plain backgrounds. An example is shown in Figure 2.4. This visual style is used for a variety of purposes, such as comics, novel illustrations, technical drawings and also animation frames of cartoons. Since line art intentionally attempts to depict objects using simple primitives, it is uniquely suited to be represented using vector formats.

2.1.3 Vectorizaton

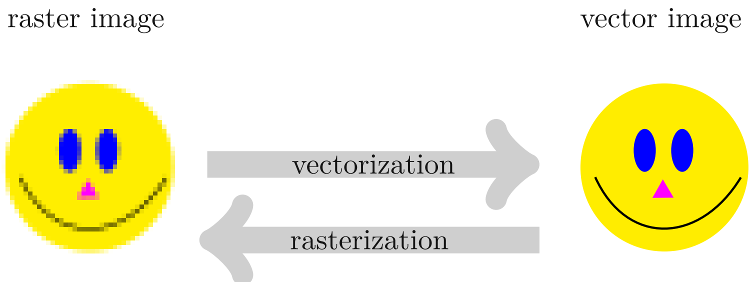


Figure 2.5: The process of vectorization and rasterization.

Image vectorization - also referred to as *image tracing* - is the process of converting a raster image to a vector image. The inverse of this process (converting a vector into a raster image) is referred to as *rasterization*. The process is depicted visually in Figure 2.5.

There exists a non-injective relation between vector images and raster images, i.e., for each vector image, there exists a corresponding raster image, but multiple vector images can have the same corresponding raster image. In turn, for each raster image, there can be multiple corresponding vector images. Therefore, it is always possible to convert a given vector image into the corresponding raster image. However, when converting a raster image into a vector image, there exists multiple (in essence, infinite) potential corresponding candidates.

Since there is a non-unique set of vector images that map to the same raster image, it is non-trivial to ascertain the quality of the vectorization result. As explained in Section 2.1.2, in order for the resulting vector image to actually benefit from the advantages of the format, the primitives and structure need to be semantically meaningful with respect to a specific usage. Traditional vectorization algorithms attempt to approximate meaningful primitives using surrogate heuristics, whereas learned algorithms attempt to exploit statistical correspondences in the training data to derive a meaningful result. In general, measuring the primitives and structure of a generated vector image is highly dependent on the intended usage of the vector image.

It is important to note that most displays used today are raster displays. Hence, in order to actually view a vector image on a display, it is necessary to rasterize the vector image. However, most viewing software does not rasterize the vector image a priori, instead, it is interpreted and lazily rendered at the zoom and position level indicated by the user. This can be experienced by zooming into a vector image in this document.

2.2 Limited Animation Production

Limited animation is an animation technique in which moving parts (or, *cels*) of a drawing are reused across multiple frames. This stands in contrast to full animation, in which every frame is completely redrawn, which leads to a more fluid animation style. Hence, in order to provide a compelling experience for viewers, limited animation has to rely more on visual *tricks* such as rich backgrounds (which mostly remain still within a scene and therefore can be reused).

Figure 2.6 gives an overview of the typical production process for hand-drawn limited animation. Note that since limited animation enjoys wide application in the Japanese animation industry, it contains both English and corresponding Japanese production terms. The process starts with the director creating a storyboard, which contains very rough sketches giving an overview of the animation sequence. Based on the storyboard, the key animator draws the layout (or, *1st key animation*) on paper. It encompasses all keyframes in a given scene (also referred to as *cut*), as well as instructions for later stages (such as camera movement). The keyframes themselves do not produce fluid animation, since they are only drawn for critical stages within an animation sequence (beginning, junctures and end). The 1st key frames already include rough sketches for the background as well as the cels. The layout is then normally checked and possibly corrected by the animation director. After the layout is approved, the rough 1st key

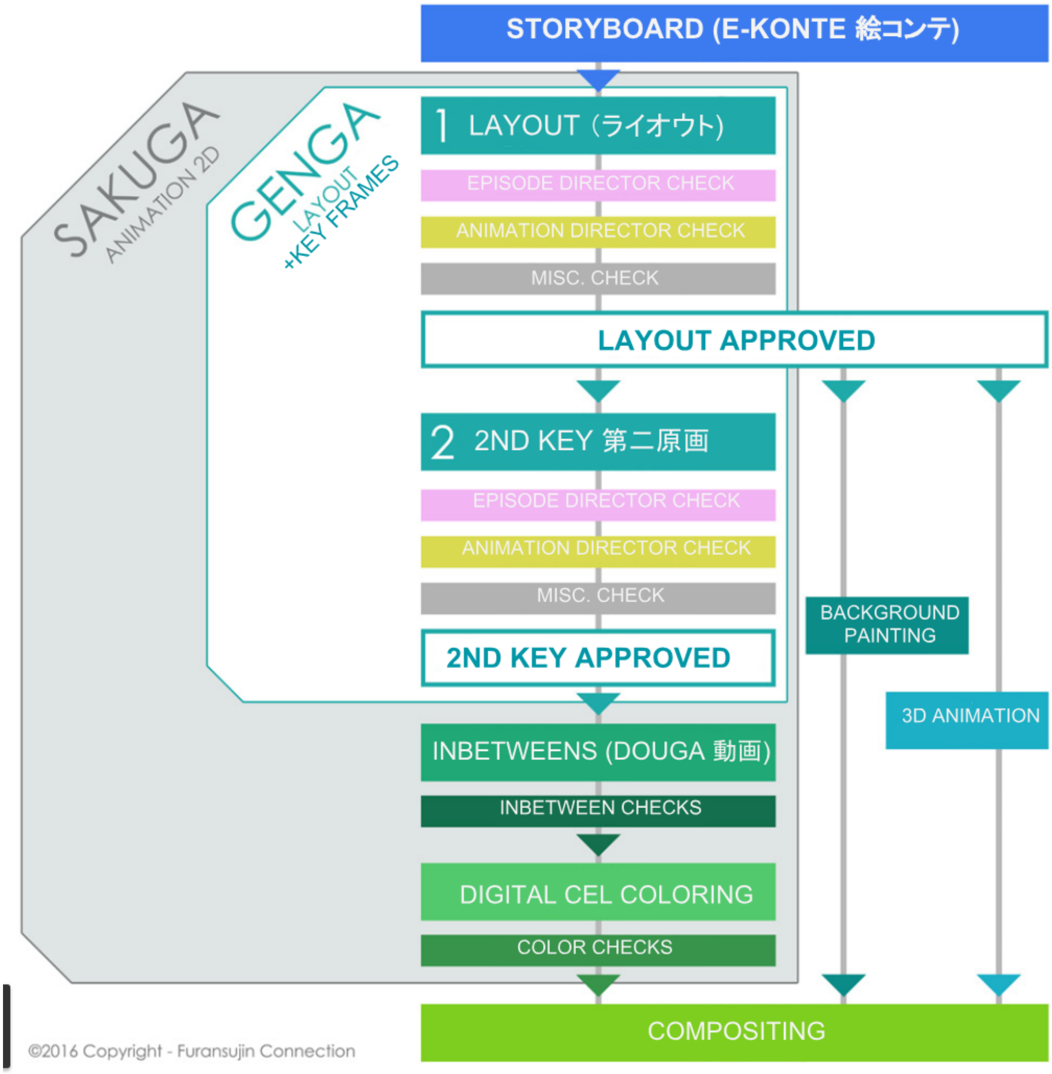


Figure 2.6: Example of a limited animation production workflow [Furansujin Connection, 2016]. The steps dealing with Sakuga (i.e., the steps following *2ND KEY APPROVED* and before *COMPOSITING*) are usually done using vector images.

frames are cleaned up while taking the animation director's corrections into account in order to produce 2nd key animation frames. This process can be repeated multiple times, until the animation director finally approves the keyframes.

Afterwards, the approved keyframes are retraced and digitized as vector images. These clean final frames are not full drawings, but include the outlines of each object in a scene, decorative lines, as well as lines indicating shadows, lighting and color regions. It can be interpreted as a semantic description of the essential parts (mostly cels) of the final

(a) A final composed animation frame by Uchida et al. [2019], which is in raster format by default.

(b) The corresponding re-imagined clean animation keyframe in raster format provided by Tonari Animation.

Figure 2.7: An example of a clean animation keyframe and the corresponding final animation frame.

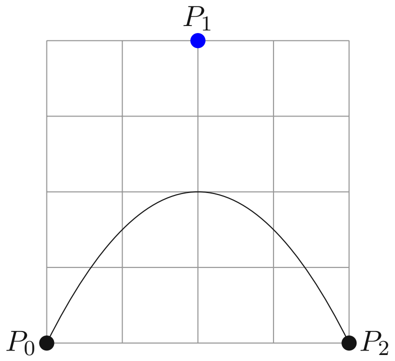
frame. An example can be seen in Figure 2.7b.

The keyframes are retraced as vector images since this format makes it easy to apply succeeding steps in the workflow. Vector images can be easily and accurately edited by simply altering primitives. Furthermore, images are often worked on at different scales, where the resolution-independent nature of vector images is beneficial. This is especially important when working on tiny details of the image.

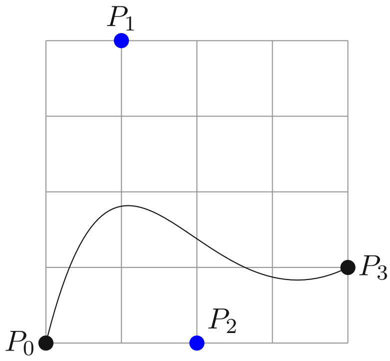
Figure 2.8: Three successive clean frames of an animation scene in vector format provided by Tonari Animation. If rapidly shown in succession, the person appears to be moving. Notice minor differences among them, such as the angle of the pencil or the location of strains of hair.

While up to now only keyframes were drawn, in order to produce fluid movement between them, *inbetweens* are created using the clean keyframes as reference. This is done at such a late stage in order to speed up the process (since only keyframes need to be revised and refined in the initial steps as opposed to all frames). Inbetweens need to be drawn exactly like the keyframe, while having small deviations for moving parts. For the workflow succeeding this step, there is no distinction between keyframes and inbetweens anymore. The resulting images are referred to by various terms interchangeably; most consistently used is the term *douga* (動画, lit. *moving image*) in Japanese limited animation production. Figure 2.8 depicts three examples of *douga*.

After the clean frames are drawn and approved, they are colored according to their color indications. The color indications in the key animations serve to retain temporal consistency for the coloring. Nowadays, this is done digitally using bucket filling, i.e., changing the color within a closed area in the image. The precise nature of the vector image makes this process easier, since there is no ambiguity where exactly color areas are located. Since the resulting animation video will be displayed on raster displays at a fixed resolution, for succeeding steps it is not necessary for the frames to be in vector format anymore. Hence, they are rasterized and composited with the background, which was drawn in parallel in raster format. While in theory, clean frames could be rasterized at any arbitrary size, usually, they are rasterized with a width of 720 pixels. Finally, camera movement, special effects, 3-D animation and small fixes are added. An example of such a final frame is depicted in Figure 2.7a.



(a) A quadratic bezier curve, defined by a start point P_0, an end point P_2 and one control point P_1.



(b) A cubic bezier curve, defined by a start point P_0, an end point P_3 and two control points P_1 and P_2.

Figure 2.9: Examples of bezier curves.

2.2.1 Clean Frames

The focus of this work are the clean frames as used in the production workflow after the 2nd key animation and prior to the coloring, as shown in Figure 2.7b. Digital clean frames are primarily drawn using *bezier curves* [de Faget de Casteljau, 1986] as graphical primitive. Bezier curves are smooth parametric functions that approximate real-world strokes. Figure 2.9 gives two examples of bezier curves. They are parameterized by a start and an end point, as well as a variable number of control points. The curve is then constructed using a combination of linear interpolations between the points. The number of control points indicates the degree of the curve; most commonly used are *quadratic* and *cubic* bezier curves, as nearly all relevant shapes can be represented using them. Note, that a bezier curve of degree n can also be represented by a bezier curve of degree $m > n$. Furthermore, every bezier curve can be split at any point into two bezier curves of the same degree.

Clean frames can be considered a type of line art, specifically defined by the following features.

- Constant stroke width: Clean frames are drawn using mostly quadratic and cubic bezier curves with a constant stroke width. While line art usually uses variable stroke widths, this is not a common style for limited animation, specifically in Japanese productions.

- High number of curves: On average, the amount of curves per image over 1000 (see Section 4.2.1), which is significantly more than the amount of curves used for simple line art that is often the target for automatic vectorization (such as doodles, fonts or cartoons).

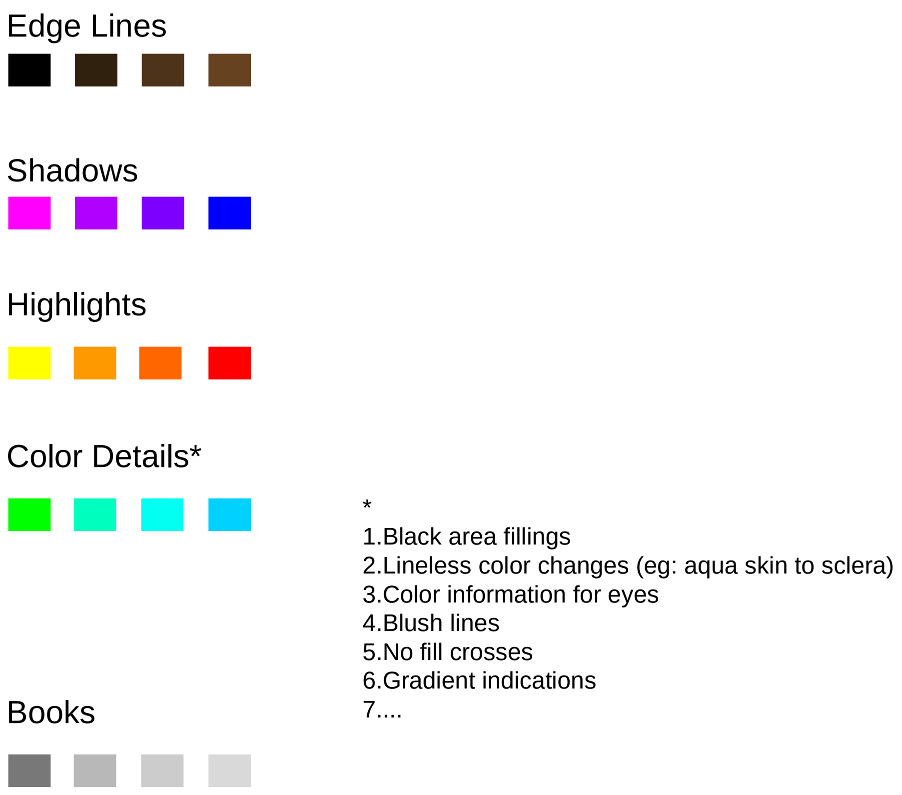


Figure 2.10: Our proposed standardized keyframe color schema, which is used in the dataset [Kugler, 2023].

- Semantically meaningful colors: The color of each curve carries semantic meaning, namely as indication for succeeding steps in the animation pipeline. There is no industry-standard for the color scheme, with studios mostly opting to use their proprietary schemes. The scheme used by example images in this work (all provided by Tonari Animation) is defined in Figure 2.10. It can be observed in Figure 2.7. For example, note how the regions enclosed by blue lines are colored darker than the ones enclosed by red lines.

- Exact boundaries: In order to speed up the colorization step of the production workflow, it is necessary to ensure that all color regions are sufficiently enclosed by curves. This way, coloring can be trivially accomplished using bucket filling.

Even if clean frames are drawn as vector images, they might sometimes only be available in raster format. This could be the case of older clean frames, different studios working on different stages of the workflow or if it is necessary to export the image from the drawing program to another program. In this case, they need to be manually retraced before inbetweens are created, which can take around an hour. Hence, there is a need to have an automatic way of vectorizing them.

Figure 2.11: The essential information of a clean animation keyframe. In essence, Figure 2.7b with all filled color regions removed.

The objective of this work is to create an automatic way of vectorizing clean frames. For the vectorization, it is not necessary to handle the filled color regions, since those can be easily restored using pre- or post-processing and, failing that, can be trivially added manually. The essential part are the lines and curves themselves, as shown in Figure 2.11.

Finally, in order for the vectorization method to be practically suitable, it is important that the result minimizes the amount of manual fixing required. Hence, it is more important that the restored lines are correct, even at the expense of not all lines in the input image being covered.

2.3 Deep Learning

When attempting to tackle a task computationally, it is common to explicitly define an algorithm that solves the task in a structured way with manually set parameters. In practice, this approach is often unfeasible for complex tasks with high-dimensional data, such as image generation or speech recognition. A different approach is *machine learning*, i.e., to assemble a dataset of correct inputs and outputs and defining an algorithm which exploits statistical correlations in the dataset to fit a model that generalizes to data outside of this dataset. *Deep learning* is a subset of machine learning that has gained traction in recent years due to practical successes [Ciresan et al., 2012, Krizhevsky et al., 2012, Silver et al., 2016, Devlin et al., 2019, Brown et al., 2020, Rombach et al., 2022, Ouyang et al., 2022].

The basis of deep learning are *neural networks* [Rosenblatt, 1958], which are explained in Section 2.3.1. Different ways of formulating objectives for and measuring neural networks are explored in Section 2.3.2. Section 2.3.3 describes Convolutional Neural Networks (CNNs) [Fukushima and Miyake, 1982, LeCun et al., 1989], which are a type of neural networks suited for spatial data such as raster images. The equivalent for sequential data, Recurrent Neural Networks (RNNs) [Rumelhart et al., 1986], are detailed in Section 2.3.4. The Transformer architecture [Vaswani et al., 2017, Schmidhuber, 1992], a recently developed model architecture attempting to combine the advantages of CNNs and RNNs is introduced in Section 2.3.5. Section 2.3.6 describes a flexible framework for generative models called encoder-decoder framework [Sutskever et al., 2014, Mikolov, 2012].

2.3.1 Neural Networks

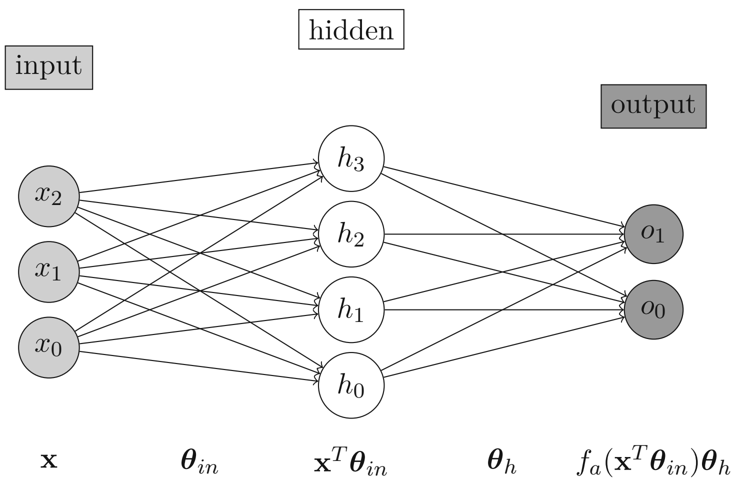


Figure 2.12: Architecture of a multi-layered perceptron (MLP). Circles of same color represent the variables within a layer, while the connections between circles indicates weights. The mathematical representation is given below.

Consider a set of input data $\mathbf{x} \in \mathbb{R}^m$ and a set of outputs $\mathbf{y} \in \mathbb{R}^l$ of unspecified dimensions, which are related by the ground truth function $F(\mathbf{x}) = \mathbf{y}, F : \mathbb{R}^m \to \mathbb{R}^l$. In the context of *machine learning*, $\mathbf{x}$ can be referred to as *predictors* and $\mathbf{y}$ as the *response* or *label*. For example, in the case of binary image classification the predictor could be a sequence of the color values of all pixels in an image, while the response could be a dichotomous variable indicating whether the image is part of the positive class. The objective of a machine learning algorithm is to fit a model f that approximates F using freely changeable (or, *learnable*) parameters $\boldsymbol{\theta}$, i.e., $f(\mathbf{x}; \boldsymbol{\theta}) \approx F(\mathbf{x})$.

Neural networks [Rosenblatt, 1958] can be viewed as special, nonlinear case of the approximation f. Specifically, a neural network consists of three types of layers:

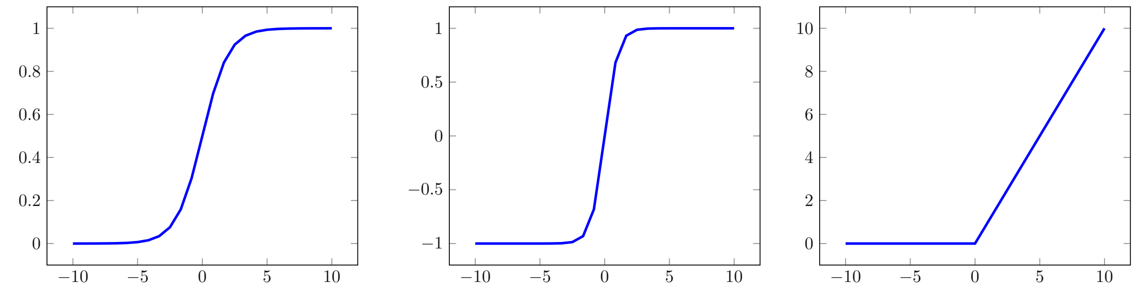
- One input layer $\mathbf{x}$
- n hidden layers and
- one output layer $\mathbf{o}$

Intuitively, each hidden layer learns to detect important features of the preceding layer. These features are then used in the last layer - the output layer - to compute the relevant, task dependent outcome.

The standard feedforward neural network, the MLP, is depicted in Figure 2.12. Note that Figure 2.12 depicts a neural network with only one hidden layer. Theoretically, this 1-layered MLP is able to approximate any function [Hornik et al., 1989, Cybenko, 1989]. However, in practice, it has been shown that neural networks that consist of many hidden layers perform better on complex functions [Amari, 1967, Safran and Shamir, 2017, Petersen and Voigtländer, 2018]. The amount of hidden layers is also referred to as the *depth* of the neural network, leading to the term *deep learning* [Dechter, 1986, Chen et al., 2001].

In the MLP architecture, each hidden layer consists of a predetermined size k of nodes. Every node is connected to all nodes of the preceding layer (hence also called *fully connected*). Each of these connections is assigned a learnable weight θ. Thus, the value of each node is the weighted sum of all nodes of the preceding layer. Therefore, each layer in a neural network can be described as inner product $\mathbf{h} = \mathbf{x}^T \boldsymbol{\theta}$, where $\mathbf{x}$ is the vector of nodes of the preceding layer and $\boldsymbol{\theta} \in \mathbb{R}^{k \times |\mathbf{x}|}$ is a matrix, where $\boldsymbol{\theta}_i \forall i \in [0, k]$ is a vector of weights for one node. For brevity, the bias term is omitted. Intuitively, each node is a specific *representation* of the preceding layer which assigns a certain importance (i.e., weight) to every node of the preceding layer.

After the node values of the hidden layer are computed, an *activation function* $\phi(x)$ is applied on them. Early MLPs were inspired by biological neural networks and used the sigmoid activation function defined in Equation (2.1). This function is depicted in Figure 2.13a. A similar activation function is the tanh activation function defined in Equation (2.2) and depicted in Figure 2.13b. Contemporary neural networks mainly use the Rectified Linear Unit (ReLU) activation function [Nair and Hinton, 2010] defined in Equation (2.3) and depicted in Figure 2.13c.



(a) Sigmoid function. (b) Tanh activation function. (c) ReLU activation function.

Figure 2.13: Graphs depicting activation functions.

$$\sigma(x) = 1/(1 + \exp(-x)) \tag{2.1}$$

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.2}$$

$$\mathrm{relu}(x) = \max(0, x) \tag{2.3}$$

The output of each hidden layer is thus $\phi(\mathbf{h}) = \phi(\mathbf{x}^T\boldsymbol{\theta})$. Notice that ϕ serves as nonlinearity. If ϕ was not used, the output of an n-layered neural network could be computed using $\mathbf{o} = \mathbf{x}^T \prod_{i=0}^{n} \boldsymbol{\theta}_i = \mathbf{x}^T\boldsymbol{\theta}$. This is equivalent to the multiple linear regression model $\mathbf{y} = \mathbf{x}^T\boldsymbol{\beta} + \boldsymbol{\epsilon}$.

With the activation function $\phi(x)$, a two-layered MLP is defined by Equation (2.4).

$$\mathbf{o} = \phi(\phi(\mathbf{x}^T\boldsymbol{\theta}_{\mathrm{in}})\boldsymbol{\theta}_{\mathrm{h}}) \tag{2.4}$$

In Equation (2.4) $\boldsymbol{\theta}_{\mathrm{in}}$ are the connection weights between the input and the hidden layer and $\boldsymbol{\theta}_{\mathrm{h}}$ are the connection weights between the hidden and the output layer.

Optimization

In order to approximate the ground truth function $F(\mathbf{x})$ using the model $f(\mathbf{x};\boldsymbol{\theta})$, the optimal parameters $\boldsymbol{\theta}^\star$ have to be found. In machine learning parlance, this is referred to as *fitting*, or *training* the model. In the case of neural networks, this means finding the optimal parameters $\boldsymbol{\theta}_i^\star$ for each hidden layer, such that $f(\mathbf{x};\boldsymbol{\theta}^\star) \approx F(\mathbf{x})$. Historically, fitting neural networks often failed to converge to optimal parameters [Minsky and Papert, 1969]. However, this changed with the introduction of the backpropagation algorithm (also referred to as reverse-mode differentiation) [Dreyfus, 1962, Linnainmaa, 1970], which made it possible to efficiently train neural networks.

The first step in optimization is *weight initialization*, i.e., to initialize the learnable parameters $\boldsymbol{\theta}_i$ with random numbers. Next, the output of the neural network is computed as $\mathbf{o} = f(\mathbf{x}; \boldsymbol{\theta})$. In order to evaluate the output of the network, a *loss function* is defined as $L(\mathbf{o}, \mathbf{y})$. L calculates the error between $\mathbf{o}$ and the ground truth $\mathbf{y}$. It is important to choose a proper loss function fitting the specific task. A simple loss function sometimes used in image generation is the mean squared error, or L^2 distance, defined by Equation (2.5).

$$L^2(\mathbf{o}, \mathbf{y}) = \frac{\sum_{i=1}^{|\mathbf{o}|}(o_i - y_i)^2}{|\mathbf{o}|} \tag{2.5}$$

Using the Stochastic Gradient Descent (SGD) algorithm, the loss function L can be used to *guide* the training process towards optimal parameters. This is done by computing the gradients $\partial L/\partial \boldsymbol{\theta}_i$ for each weight parameter using backpropagation. These gradients indicate how the loss L changes when $\boldsymbol{\theta}_i$ is increased by an infinitesimal amount. As the loss function needs to be *minimized*, the parameters are updated using the SGD update step defined by Equation (2.6).

$$\boldsymbol{\theta}_i = \boldsymbol{\theta}_i - \eta \frac{\partial L}{\partial \boldsymbol{\theta}_i} \tag{2.6}$$

η in Equation (2.6) is a hyperparameter called *learning rate* or *step size*. It controls the magnitude of the weight parameter updates and is often a small positive number, as higher numbers quickly lead to exploding or vanishing gradients [Hochreiter, 1991].

The SGD update step, i.e., computing the outputs and changing the parameters is repeated until a set of parameters that results in a sufficiently small loss is found. In recent times, several enhancements to the SGD algorithm were proposed. Most of them are *adaptive* optimization algorithms, i.e., they adapt the learning rate η according to the previous gradients or parameter updates. Chief among them are Adam [Kingma and Ba, 2015], AdamW [Loshchilov and Hutter, 2019] and RMSprop [Hinton et al., 2014]. While it is still unproven that these optimization algorithms are strictly better than SGD [Hardt et al., 2016, Wilson et al., 2017], they do perform better on some tasks and model architectures [Liu et al., 2020, Anil et al., 2019].

For the optimization process to work, it is crucial that the gradients required for the update step $\partial L/\partial \boldsymbol{\theta}_i$ can actually be computed. This means that the loss function $L(\mathbf{o}, \mathbf{y})$ needs to be *differentiable*, i.e., there must be a derivative at all points in its domain. Since $\mathbf{o} = f(\mathbf{x}; \boldsymbol{\theta})$ and since the gradients are computed directly with respect to individual $\boldsymbol{\theta}_i \in \boldsymbol{\theta}$, it follows that the model f itself also needs to be differentiable. As an aside, the ReLU activation function defined in Equation (2.3) is actually not differentiable at $x = 0$. In order to enable it to be used in the model anyways, the derivative at $x = 0$ is arbitrarily set to 0 ($\text{relu}'(0) = 0$).

Regularization

A common occurrence while training neural networks is *overfitting*, i.e., finding parameters $\boldsymbol{\theta}$ that closely fit the data used to train the network, but do not generalize to unseen data. In order to prevent this, several regularization schemes are employed.

Model and Dataset Size In general, the larger a model is (i.e., the more freely changeable parameters θ it contains) in relation to the dataset size, the easier it is to overfit. Therefore, an important design decision to reduce overfitting is to either reduce the model parameters to the smallest capacity required to converge or to increase the dataset size.

Weight initialization Weight initialization is an important decision for the training process, as wrongly initialized neural networks do not converge to a sufficient result [Glorot and Bengio, 2010, Sutskever et al., 2013]. Therefore, one often draws the initial weights of each hidden layer from a Gaussian distribution $\boldsymbol{\theta}_i \sim \mathcal{N}(0, 1)$ and scales the variance with a constant. One commonly used method is Xavier initialization [Glorot and Bengio, 2010], which scales the random numbers by $2/(n_{\text{pre}} + n_{\text{next}})$, where n_{pre} is the number of nodes of the preceding layer and n_{next} is the number of nodes in the succeeding layer. Another initialization scheme specifically designed for ReLU neural networks is He initialization [He et al., 2015] with a scale factor of $\sqrt{2/n}$, where n is the number of nodes of each hidden layer.

Batch normalization Neural networks are normally fitted by processing subsets of the training dataset - *batches* - in parallel. This is primarily done to utilize the nature of Graphics Processing Units (GPUs) in order to speed up the training process. However, it is also possible to use batch statistics to normalize individual observations with respect to the batch. This *batch normalization* [Ioffe and Szegedy, 2015] ensures that batches are normally distributed, which might improve the stability of the training process. The authors claim that internal covariate shift - i.e., differing distributions of intermediate layer outputs - are detrimental to convergence. However, there exist alternative explanations for how batch normalization actually brings about its regularizing effects in practise [Santurkar et al., 2018, Yang et al., 2019, Kohler et al., 2019].

More specifically, given a batch $\mathbf{x}_B \subset \mathbf{x}$, each observation $x_i \in \mathbf{x}_B$ is centered using the batch mean $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$ and scaled by the batch variance $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$ as given by Equation (2.7). In this equation, k denotes the dimension, since batch normalization is applied per-dimension.

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\left(\sigma_B^{(k)}\right)^2 + \epsilon}} \tag{2.7}$$

Dropout Another way to regularize neural networks and reduce overfitting is to randomly set individual nodes in hidden layers to 0. This technique is called *dropout* [Hinton et al., 2012] and makes the model more robust with respect to individual nodes and their connection weights, since it can not rely on the node value being present.

Dropout is mainly applied while fitting the model. Consider the hidden layer $\mathbf{h} = \mathbf{h}_{-1}\boldsymbol{\theta}$. After computing the hidden layer nodes $\mathbf{h}$ using the preceding layer $\mathbf{h}_{-1}$ and the connection weights $\boldsymbol{\theta}$, each node $h \in \mathbf{h}$ is independently set to 0 with probability p (it is common to set $p = 0.5$). Since this is independently done for every step of the optimization process, different nodes will be "dropped" at each step. Hence, in order to maximize the optimization objective, the connection weights have to be fitted in a way that is robust to a share p of the nodes randomly being dropped (and thereby a share p of the weights not contributing to the output).

2.3.2 Metrics and Losses

Recall that a machine learning model $f(\mathbf{x};\boldsymbol{\theta}) = \mathbf{o}$ approximates a ground truth function $F(\mathbf{x}) = \mathbf{y}$ by being fitted to an input dataset $(\mathbf{x}, \mathbf{y})$ in order to produce outputs $\mathbf{o}$ that fit the ground truth output (or, response) $\mathbf{y}$. It follows, that there is a need for a single numerical value to measure this fit. There are two ways this value is calculated and used in relation to a machine learning model: firstly, to evaluate the model after it has been trained using *metrics* (often against other models or baselines) and secondly, in order to guide the training process using *loss functions*. This section introduces these related concepts.

Classification

Since metrics and losses formulate the objective, they depend on the task of the model. Machine learning tasks can be roughly divided into two categories by the nature of the response $\mathbf{y}$. While in regression tasks, $\mathbf{y}$ is continuous, in classification tasks $\mathbf{y}$ is discrete.

Regression does not impose strict assumptions on $\mathbf{y}$ is therefore the standard case introduced in Section 2.3.1. An important image-related task that roughly falls into this category is image generation, i.e., to generate output images $\mathbf{o}$ that look similar to the ground truth $\mathbf{y}$.

On the other hand, classification has more strict assumptions on $\mathbf{y}$. The task is to assign to an input $\mathbf{x}$ a predefined class out of S classes. Since machine learning models operate on vectors, the class in the ground truth is represented as a vector $\mathbf{y} \in \mathbb{R}^S$, where each element $y_i \in \mathbf{y}$ is an indicator of the class $i \in [1, S]$. If the input is assigned class s, $y_s = 1$ and $y_j = 0 \forall j \in [1, S]$ except $j = s$. Appropriately, the output of the model $\mathbf{o}$ has the same structure as the ground truth $\mathbf{y}$, except it usually consists of continuous class *probabilities* instead of categorical class indicators, i.e., whereas $y_i \in \{0, 1\}$, $o_i \in [0, 1]$. In the case of binary classification, i.e., if $S = 2$, the two class probabilities are exactly the inversion of each other and can be reduced to a single probability, i.e., in this case the class is represented by a single number.

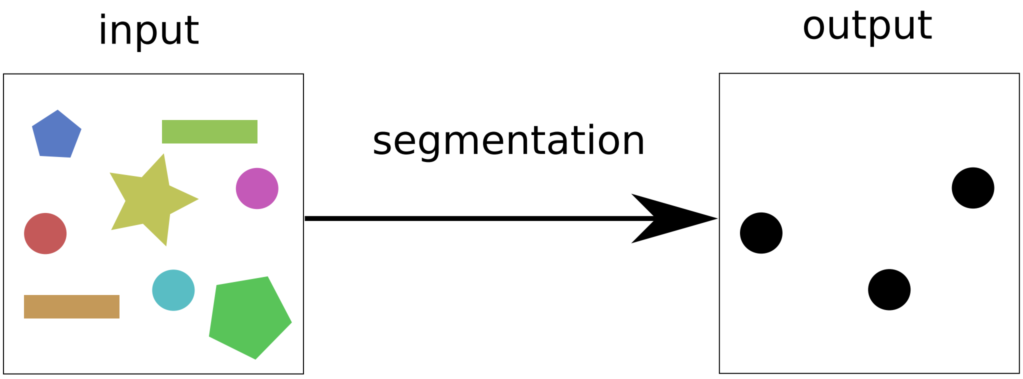


Figure 2.14: An example of image segmentation, where the circles in the image have to be segmented.

There are image-related tasks that can be framed as classification tasks. Consider the case of image segmentation, which has the objective of partitioning an image into multiple segments, where each segment belongs to one of multiple classes. All regions of a class share certain task-dependent characteristics. Figure 2.14 shows an example of an image segmentation task. This example depicts a binary image segmentation task where the image has to be segmented into two classes: segments that correspond to circles and the rest.

In other words, the output of a segmentation model assigns a class to each pixel of the input image. Given an input image $\mathbf{x} \in \mathbb{R}^{W \times H \times C}$ of width W, height H and C color values per pixel, the output is $\mathbf{o} \in \mathbb{R}^{W \times H \times S}$, where S is the number of classes. In detail, each pixel $\mathbf{x}_{i,j} \in \mathbf{x}$ at location (i, j) in the input image $\mathbf{x}$ is assigned a vector $\mathbf{o}_{i,j} \in \mathbf{o}$ of length $|\mathbf{o}_{i,j}| = S$, where the element $o_s \in \mathbf{o}_{i,j}$ indicates the probability that the pixel belongs to class s $\forall s \in [1, S]$. Furthermore, it holds that $o_s \in [0, 1] \forall s$ and $\sum_{k=0}^{|\mathbf{o}|} o_k = 1$. In the special case of $S = 2$, the class dimension of the output vector can be interpreted as a monochrome color model and the output can be visualized as an image in which the positive class is indicated as black and the negative class is indicated as white. An example of this is shown in Figure 2.15. Note that clean animation keyframes can be seen as image segmentation output of corresponding final animation frames (see Figure 2.7).

Keep in mind that image segmentation is often associated with class imbalance, i.e., under-representation of a class in the training dataset, which has to be considered by metrics and loss functions.

Metrics

It is important to evaluate how good a model f performs. This is measured using metric that computes the difference between the model output $\mathbf{o}$ and the ground truth $\mathbf{y}$.

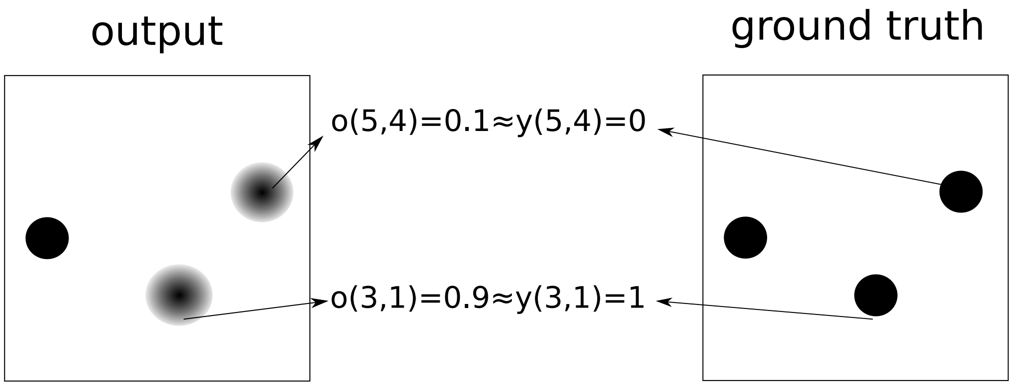


Figure 2.15: An example of binary image segmentation using binary classification. On the left, an output of a model that has not yet converged is visualized as a monochrome image. The ground truth data is displayed on the right. Note the continuous class probabilities being used in the model output.

However, notice that $\mathbf{y}$ is used to both fit the model and evaluate it. Therefore, a model that trivially memorizes the input and output data would attain a perfect evaluation score. However, this model would not generalize to unseen data. Since the primary aim of a model is to be applied to unseen data, it is conductive not to evaluate the model on these instead of the data it has been fitted to.

In order to attain evaluation on unseen data, the principal training scheme is splitting the available dataset of input data $\mathbf{x}$ and labels $\mathbf{y}$ into a training dataset $\mathbf{x}_{\text{train}} \subset \mathbf{x}, \mathbf{y}_{\text{train}} \subset \mathbf{y}$ and a test dataset $\mathbf{x}_{\text{test}} \subset \mathbf{x}, \mathbf{y}_{\text{test}} \subset \mathbf{y}$, where $\mathbf{x}_{\text{train}} \cap \mathbf{x}_{\text{test}} = \varnothing$ and $\mathbf{y}_{\text{train}} \cap \mathbf{y}_{\text{test}} = \varnothing$. The training dataset is used solely to fit the model, while the test dataset is used to evaluate the model. Special care needs to be taken such that there is no information leak from the test dataset into the training dataset.

Specifically, the metric functions introduced below are applied on the ground truth of the test dataset $\mathbf{y}_{\text{test}}$ and the model output $\mathbf{o} = f(\mathbf{x}_{\text{test}}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ has been fitted using the training dataset $\mathbf{x}_{\text{train}}, \mathbf{y}_{\text{train}}$. The metrics are introduced for regression and classification models, respectively.

Regression This section introduces two common metrics that can be used to measure models trained for regression tasks. Let $\mathbf{o}$ and $\mathbf{y}$ be in Euclidean space, then the ordinary distance between them $\mathbf{y} - \mathbf{o}$ (i.e., the error) can be directly calculated and used as metric. However, by simply computing the subtraction, large negative distances would actually contribute to minimizing the error. There are two common metrics which prevent this. While the mean absolute error (MAE) (also referred to as L^1 distance) as defined in Equation (2.8) uses the absolute of the error, the mean squared error (MSE) (or, L^2

distance) as defined in Equation (2.5) uses the square of the error. These metrics can be used as a simple way to calculate the difference between two images.

$$L^1(\mathbf{o}, \mathbf{y}) = \frac{\sum_{i=1}^{|\mathbf{o}|} |o_i - y_i|}{|\mathbf{o}|} \tag{2.8}$$

Classification This section introduces metrics for binary classification models, which are used for the binary image segmentation task explained in Section 2.3.2. Binary classifiers are often evaluated by first computing their confusion matrix, which counts all possible configurations of class assignments per input observation (i.e., pixel in image segmentation). It is displayed in Table 2.1, where TP refers to the true positives, i.e., the amount of observations to which both the model and the ground truth assign the positive class, FP refers to the false positives, i.e., the amount of observations to which the model assigns the positive and the ground truth assigns the negative class and FN refers to the false negatives, i.e., the amount of observations to which the model assigns the negative and the ground truth assigns the positive class.

Table 2.1: Confusion matrix for a binary classifier.

	$y = 1$	$y = 0$
$o = 1$	true positives (TPs)	false positives (FPs)
$o = 0$	false negatives (FNs)	true negatives (TNs)

As confusion matrices consists of four numbers, they somehow need to be reduced to a single number. There are different ways of doing that, which are described below.

An easy metric that is commonly used is accuracy as defined in Equation (2.9). Notice that $ACC \in [0, 1]$. However, this metric does not take into account the proportion of positive class observations over negative class observations in the dataset. Therefore, in the case of class imbalance, the metric is biased towards the over-represented class.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.9}$$

One strategy of combating class imbalance in binary classification problems is to consider the intersection between the output and the ground truth. This is done using the Intersection-over-Union (IoU) [Gilbert, 1884, Jaccard, 1912, Tanimoto, 1958] (also referred to as Jaccard index), which is defined in Equation (2.10). Notice that $J \in [0, 1]$. As the name suggests, IoU divides the intersection, i.e., the amount of observations to which both the output and the ground truth assign the positive class by their union. It can be seen in Equation (2.10) that only the amount of true positives increases the IoU metric. Hence, even if the negative class is over-represented, it does not affect the metric.

$$J = \frac{TP}{TP + FP + FN} \tag{2.10}$$

Another metric that is closely related to the IoU is the dice coefficient [Dice, 1945, Sørensen, 1948] (also referred to as Sørensen index or F_1 score), which is defined in Equation (2.11). Notice that $S \in [0, 1]$. It differs from the IoU by multiplying the true positives by 2. This way, the true positives contribute exactly the same as the false positives and false negatives together to the denominator. Aside from that, it is identical to the IoU and related by $S = 2J/(1 + J)$. Hence, the IoU and dice coefficient are positively correlated.

$$S = \frac{2TP}{2TP + FP + FN} \tag{2.11}$$

Losses

As explained in Section 2.3.1, loss functions are an integral component of the optimization process of neural networks. This section will detail important loss functions used for tasks involving raster images. A loss function $L(\mathbf{o}, \mathbf{y})$ is the mathematical representation of the optimization objective (i.e., the task). It assesses the fit of the model to the training dataset by computing the numerical error between the neural network output $\mathbf{o}$ and the ground truth $\mathbf{y}$. The loss function is used to *guide* the model to convergence (i.e., optimal performance). Since neural networks are optimized by adjusting the model parameters $\boldsymbol{\theta}$ using SGD, it needs to be a continuous function, i.e., small changes of the argument should not lead to large changes of the image. This implies that the function is differentiable at all elements in its domain.

In general, it is possible to optimize neural networks using simple metrics such as accuracy (explained further in Section 2.3.2). However, in most cases, there exists a loss function that encapsulates the same meaning, but satisfies the above properties, leading to better convergence performance by the model.

Again, as is the case for the above metrics, loss functions are introduced first for regression and subsequently for classification tasks.

Regression The regression metrics MAE and MSE as defined in Equations (2.5) and (2.8) can be directly used as a loss functions, since they are continuous. However, both have their own advantages and disadvantages. Since the MSE amplifies values, it is not as robust to outliers as the MAE. On the other hand, it is smoother at values near 0. In order to combine the advantages of both losses, Huber [1964] introduced the Huber loss, which is defined in Equation (2.12). A hyperparameter $\delta \in R^+$ serves as threshold. If the absolute error is below this threshold, the squared error is used, otherwise the absolute error is used.

$$
\begin{aligned}
\text{Huber}_i(o,y) &= \begin{cases} \frac{1}{2}(y-o)^2 & |y-o| \leq \delta \\ \delta * \left(|y-o| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \\
\text{Huber}(\mathbf{o},\mathbf{y}) &= \frac{\sum_{i=0}^{|\mathbf{o}|} \text{Huber}_i(o,y_i)}{|\mathbf{o}|}
\end{aligned}
\tag{2.12}
$$

The MAE, MSE and Huber losses measure the similarity between images at a pixel level. However, this assumes that every pixel contributes uniformly to the similarity, which is often not the case. Furthermore, it has been posited that Euclidean losses suffer from the *curse of dimensionality* [Zimek et al., 2012], i.e., are not meaningful for high-dimensional data such as images. A different method of measuring the similarity is to calculate the difference of high-level *features* of the images. Johnson et al. [2016] introduce the *perceptual* loss function, which implements this principle. They first process both the output and the ground truth image using a pre-trained deep neural network such as a ResNet trained on the ImageNet dataset (see Section 2.3.3) to extract high level features from the ultimate hidden layer. Then, the loss function is simply the difference between these feature vectors.

Classification

Recall that binary classification tasks such as image segmentation are often associated with class imbalance, i.e., under-representation of a class in the training dataset. If this is not considered in the loss function, the optimization process often converges to the local optimum of simply ignoring this class.

Cross-entropy Loss The cross-entropy loss is a popular loss function for classification tasks. It is defined for S classes in Equation (2.13) and is related to the softmax function defined in Equation (2.27) (see Section 2.3.5). Note that k is the assigned class in the ground truth, i.e., for which $y_k = 1$. Intuitively, the loss function interprets the individual elements of $\mathbf{o}$ as unnormalized logarithmic class probabilities and calculates their difference to the ground truth class indications $\mathbf{y}$. That is, $CE(\mathbf{o},\mathbf{y})$ is minimized if $o_k = 1$ and $o_i = 0\ \forall i \in [1,S]$ given that $i \neq k$. Equation (2.13) contains the term $w_s \in [0,1]$, which weights the distance per class. The standard cross-entropy loss is unweighted, i.e., $w_s = 1 \forall s \in [1,S]$. In the case of class imbalance, it makes sense to balance the contribution of each class to the loss by increasing or decreasing their weight respective to their representation. In this case $w_s \neq 1$ for at least one $s \in [1,S]$ and the loss is referred to as *weighted* cross-entropy loss. In case of binary classification, *binary* cross-entropy loss as defined in Equation (2.14) can be used.

$$
CE_S(\mathbf{o},\mathbf{y}) = -w_k \log\left(\frac{\exp(o_k)}{\sum_{j=1,j\neq k}^{S} \exp(o_j)}\right) \tag{2.13}
$$

$$BCE(o, y) = CE_2(o, y) = \begin{cases} -w \log(o) & \text{if } y = 1 \\ -w \log(1 - o) & \text{otherwise} \end{cases} \tag{2.14}$$

Focal loss The weighted binary cross-entropy loss defined in Equation (2.14) combats class imbalance in binary image segmentation tasks by weighing predictions of the positive class over the negative class. However, Lin et al. [2017] proposed that weighing incorrect predictions over correct predictions leads to better results. This is implemented as a modification to the binary cross-entropy loss called *focal loss*. Focal loss is defined by Equation (2.15), where $\gamma \in [0, \infty)$ is the *focusing* parameter and $\alpha \in [0, \infty)$ is the *balancing* parameter. While the balancing parameter α adjusts the weight by which correct predictions are down-weighted, the focusing parameter γ adjusts the rate at which correct predictions are down-weighted. Note that if $\gamma = 0$ or $\alpha = 0$, $FL(o, y) = BCE(o, y)$.

$$FL(o, y) = \begin{cases} -\alpha(1 - o)^\gamma \log(o) & \text{if } y = 1 \\ -(o)^\gamma \log(1 - o) & \text{otherwise} \end{cases} \tag{2.15}$$

Dice Loss A different strategy of combating class imbalance in binary image segmentation problems is to measure the overlap between the output and the ground truth, as is done in the IoU (see Equation (2.10) and the dice coefficient (see Equation (2.11)). However, these functions are not continuous. [Milletari et al., 2016] introduced a differentiable version of the dice coefficient which is defined in Equation (2.16). This dice loss function is unaffected by class imbalance by design. One disadvantage is that it is not as smooth as the binary-crossentropy loss function.

$$\text{dice}(o, y) = 1 - \frac{2yo + 1}{y + o + 1} \tag{2.16}$$

2.3.3 Convolutional Neural Networks

The MLP architecture introduced in Section 2.3.1 is designed for general-purpose input data, i.e., it does not impose strict assumptions on the input data. In contrast, the CNN architecture [Fukushima and Miyake, 1982, LeCun et al., 1989] is designed for spatial data such as raster images.

A raster image could be represented as a flattened vector containing a sequence of color values of all pixels $\mathbf{x} \in \mathbb{R}^D$, where D is the product of the image width W, image height H, and number of color values per pixel C. This way, raster images can be used as input data for MLPs. However, these input vectors quickly get very large. Consider a small raster image of width $W = 224$ and height $H = 224$ containing C color values per pixel (with $C = 3$ under the usual assumption of using the RGB color model. The input vector representing this image would consist of $W \times H \times C = 224 \times 224 \times 3 = 150528$ entries. Now, since the MLP architecture consists of *fully connected* layers, the nodes of the first hidden layer need to be connected to every element in the input vector. Assuming a hidden layer size of 1000, the vector of connection weights would consist of $224 \times 224 \times 3 \times 1000 = 150528000$ elements. This is disadvantageous for two reasons. First, the storage space required for the vector would be very large. Assuming that every entry is stored using a 32-bit floating point value, this would amount to a storage size of $150528000x32 = 602112000\text{B} \approx 602\text{MB}$ for one layer alone. Secondly, and more importantly, due to the sheer number of freely changeable parameters, the model is very hard to optimize and likely to overfit.

The amount of parameters can be reduced by replacing fully connected layers with *locally connected* layers, i.e., hidden layers in which nodes are not connected to all input nodes, but just to a smaller subset of input nodes. This could be done naively by just splitting the input layer into mutually exclusive subsets and connecting hidden layer nodes to a specific subset. However, in CNNs, the locally connected layers are constructed by taking the spatial structure of raster image data into account. This is done first by arranging raster images in a 3-dimensional grid according to the image width, image height and the number of color values (also called *channels*) $\mathbf{x} \in \mathbb{R}^{W \times H \times C}$. Next, the assumption is made that pixels that are spatially close are related to each other. Thus, each hidden layer node is connected to a rectangular (usually square) subregion of nearby pixels in the input vector (i.e., the raster image). This subregion is also called *receptive field* of the hidden layer node and is usually quite small with respect to the width and height, an example being a width and height of 3 pixels, although there are indications that larger receptive fields perform better in practice [Liu et al., 2021, 2022]. It is important to note that while the receptive field is locally connected in the spatial dimensions (i.e., width and height respectively), it is fully connected along the non-spatial dimension (i.e., the channels, in this case the number of color values per pixel).

The spatial structure of input images should be preserved across locally connected hidden layers. In other words, nodes with neighbouring receptive fields should be next to each other. Hence, locally connected hidden layers are arranged as 2-dimensional grid $\mathbf{h} \in \mathbb{R}^{W_h \times H_h}$, where W_h is the width and H_h is the height of the hidden layer. If padding is used, $W_h = W_{h-1} = W$ and $H_h = H_{h-1} = H$, i.e., the width and height remain the same across layers. Every node $n_{i,j} \in \mathbf{h}$ at position (i, j) in a locally connected layer computes $n_{i,j} = \mathbf{x}^{[i:i+c,j:j+c]}\boldsymbol{\theta}_{i,j}$ where $\mathbf{x}$ is the whole input matrix, c is the receptive field size (e.g. $c = 3$) and $\boldsymbol{\theta}_{i,j} \in \mathbb{R}^{c \times c \times C}$ are the connection weights (i.e., parameters) of the node. Put another way, the node computes higher-level features from the low-level input data of its receptive field. In this way, it is a differentiable feature extractor. Since it is differentiable, it can be used to *learn* feature extraction using the optimization approach described in Section 2.3.1. This approach is also called *representation learning*. Since $\boldsymbol{\theta}_{i,j} \neq \boldsymbol{\theta}_{k,l} \forall i, j, k, l$, each node in this architecture would learn a different feature extractor. However, the feature extraction process itself should be invariant to the location of the receptive field. As an example, a potential feature extraction task could be to detect whether an object is present in the receptive field. It is clear that the object will certainly not be in the same location in every image. In this case, every node should perform this object detection on its receptive field. Therefore, all nodes need to perform the same feature extraction. This is ensured by having the weights be shared across nodes, i.e., $\boldsymbol{\theta}_{i,j} = \boldsymbol{\theta}_{k,l} = \boldsymbol{\theta} \forall i, j, k, l$. Weight sharing also drastically reduces the amount of freely changeable parameters. Since $\boldsymbol{\theta} \in \mathbb{R}^{c \times c \times C}$, such a layer requires only $C * c^2$ connection weights. Given a receptive field size of $c = 3$ and number of colors per pixel $C = 3$, this adds up to 27 weights, which is drastically lower than the 150528000 weights required for a fully connected layer.

$$\mathbf{o}_{i,j} = \sum_{k=0}^{c} \sum_{l=0}^{c} \mathbf{w}_{k,l} \mathbf{x}_{i+k,j+l} \tag{2.17}$$

The nodes of locally connected layers with shared parameters perform a computation that is similar to discrete convolution, which is defined in Equation (2.17) and visually represented in Figure 2.16. In Equation (2.17) $\mathbf{o}_{i,j}$ is the element at position (i, j) of the output feature map $\mathbf{o} \in \mathbb{R}^{W_o \times H_o}$ (with W_o denoting the width and H_o the height), c is the receptive field size, $\mathbf{x}_{i+k,j+l}$ is the element at position $(i + k, j + l)$ of the input map $\mathbf{x} \in \mathbb{R}^{W \times H \times C}$, and $\mathbf{w}_{k,l}$ is the element at position (k, l) of the kernel $\mathbf{w} \in \mathbb{R}^{c \times c \times C}$. Intuitively, the kernel is a linear transformation encoded as a matrix, which is moved across the image width-first and used to successively compute matrix multiplications with (i.e., apply a linearly transformation on) the corresponding subregions of the input matrix. In other words, the linear transformation is *convolved* with the input matrix. Due to the similarity to convolution, the aforementioned locally connected layers are called *convolutional* layers. The important difference is that in convolutional layers, the kernel consists of parameters that are optimized during the learning procedure, i.e., $\mathbf{w} = \boldsymbol{\theta}$.

Convolutional layers as introduced above share their parameters across all nodes. Hence, only one feature extraction task can be performed per layer. In order to perform multiple feature extractions per layer, the layer is simply replicated multiple times along a non-spatial dimension. Recall that a convolutional layer is arranged as a 2-dimensional grid $\mathbf{h} \in \mathbb{R}^{W_h \times H_h}$. This layer is now replicated C_h times in order to create a 3-dimensional grid $\mathbf{h} \in \mathbb{R}^{W_h \times H_h \times C_h}$. The parameters are only shared across the spatial dimesions (i.e., width and height), but not across the channel dimension. Due to this replication, given a receptive field size c and input matrix channel size C, the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{c \times c \times C}$ extends to $\boldsymbol{\theta} \in \mathbb{R}^{c \times c \times C \times C_h}$. Hence, such a layer requires $c^2 * C * C_h$ weights. Using the above example of $c = 3$, $C = 3$ and a reasonable $C_h = 64$, this adds up to 1728 weights, which is still considerably smaller than the 150528000 weights required for a fully connected layer. Thus, by increasing the parameters along the channel dimension, this hidden layer performs C_h different feature extraction tasks. Each feature map per channel dimension is also called a *filter*, correspondingly, C_h is referred to as the *filter size*. Furthermore, the hidden layers now possess the same dimensionality as the input layer (i.e., the input image).

Note that the input layer as well as the convolutional layers are 3-dimensional matrices. Furthermore, while the convolutional layers are always locally connected along the spatial

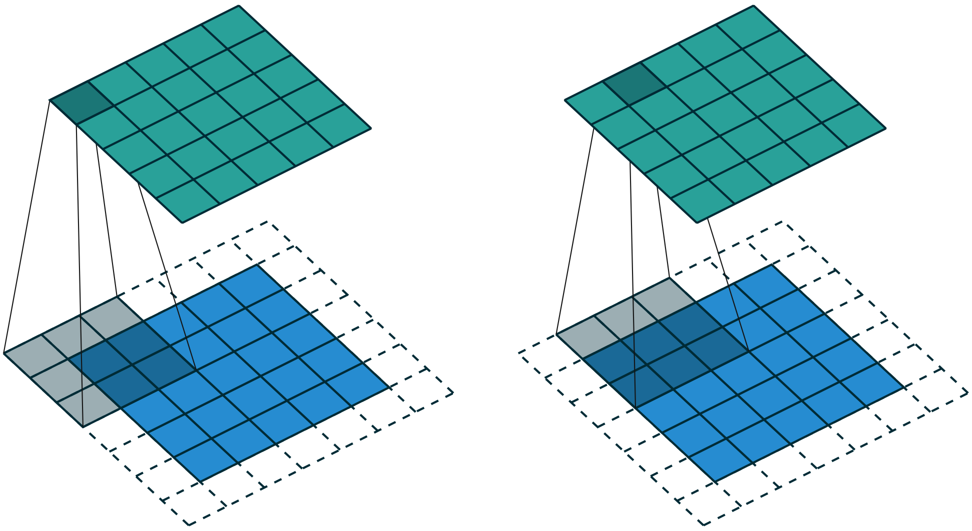


Figure 2.16: Example of a convolution with padding to preserve width and height by Dumoulin and Visin [2016]. The left image depicts the receptive field used to compute the output element at location (1,1) in the output map. The right image depicts the receptive field used to compute the output element at location (2,1).

dimensions (i.e., width and height), they are fully connected along the non-spatial dimension. This makes it trivial to use the convolutional layer output as input for a succeeding convolutional layer, i.e., to stack hidden layers. Just as in MLPs, nonlinearities are required between the layers, with the most commonly used being ReLU activations. By stacking multiple convolutional layers, it is possible to perform *hierarchical* feature extraction. While the first convolutional layer operates on low-level data (i.e., pixels), the succeeding layer operates on the extracted features of the first layer. Successively, layer n extracts features from layer $n-1$. Thus, the local features extracted in the first layers will be used to extract increasingly global features in the last layers. Finally, global features are used for downstream tasks, such as classification or object detection. Figure 2.17 gives a visual interpretation. The receptive field of nodes in each convolutional layer is quite small. However, since nodes of deeper layers operate on the output of previous nodes, they indirectly operate on the receptive field of each node in their direct receptive field. In the end, the final nodes operate on an receptive field that indirectly spans the whole image.

To summarize, the advantage of CNNs over traditional image processing is that they are *learned* hierarchical feature extractors, i.e., it is possible to fit the feature extraction on a training dataset instead of handcrafting features using heuristics. Since they are fully differentiable, they can also be finetuned to specific downstream tasks, i.e., the feature extraction process can be tailored to different tasks such as image classification, object detection and image generation. The advantage over standard MLPs is that they have

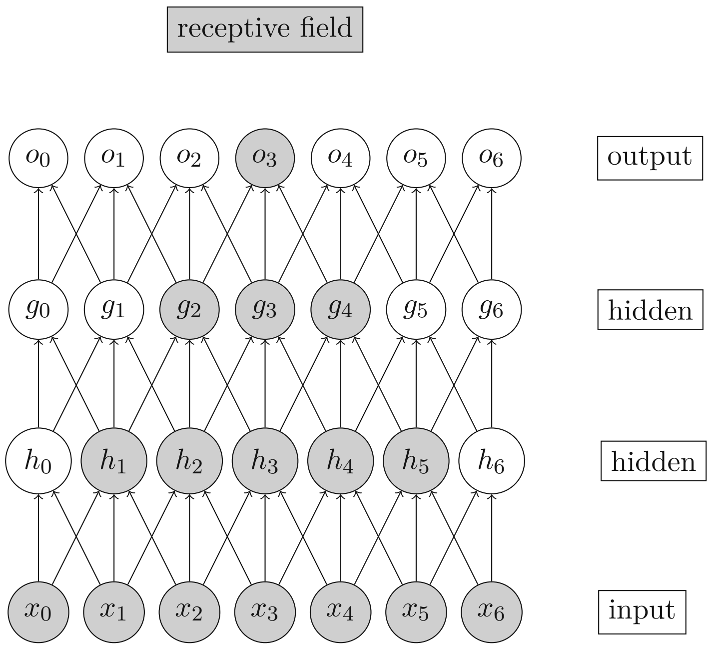


Figure 2.17: Simplified receptive field of locally connected hidden layer nodes

an inductive bias for spatial data. That means that they exploit the spatial structure of raster images by having translation-invariant feature extractors while simultaneously reducing the amount of freely changeable parameters. Furthermore, since the convolution operation can be efficiently implemented for GPUs, CNNs can be computed very quickly [Chellapilla et al., 2006, Ciresan et al., 2012, Krizhevsky et al., 2012].

Further layers

While the main component of CNNs are convolutional layers as described above, there are other layer types that are commonly used in CNNs architectures. This section gives an overview of selected CNN layers.

Pooling Recall that, if appropriate padding is used, the width and the height across layers remains the same, i.e., $W_h = W_{h-1} = W$ and $H_h = H_{h-1} = H$. For networks that are very deep, i.e., that stack a lot of convolutional layers, this fact leads to a high number of parameters and computations. In order to reduce this, *pooling layers* are inserted. These reduce the width and height of an input vector using a simple, differentiable heuristic. Chief among them is *max-pooling*, which simply reduces a rectangular (usually square) receptive field to its highest number.

Given a receptive field size c, a max-pooling layer reduces an input matrix $\mathbf{h} \in \mathbb{R}^{W_h \times H_h \times C_h}$ to $\text{max-pool}(\mathbf{h}) \in \mathbb{R}^{W_h/(c*c) \times H_h/(c*c) \times C_h}$ by simply outputting the maximum value of successive receptive fields of $c \times c$ values, i.e., where the node $n_{i,j} \in \text{max-pool}(\mathbf{h})$ at

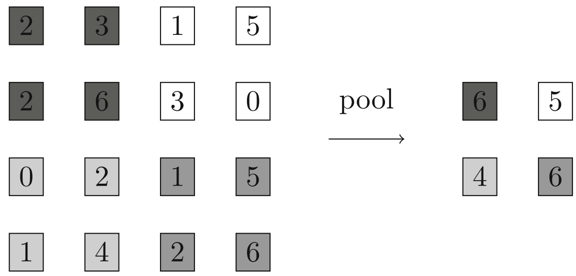


Figure 2.18: Example of a max-pooling operation.

location (i, j) is $n_{i,j} = \max(\mathbf{h}^{[i:i+c,j:j+c]})$. Note that pooling layers operate per channel, i.e., do not reduce the non-spatial dimension. A visual example disregarding the depth dimension is given in Figure 2.18. Another pooling operation that is frequently used is *average*-pooling, which computes the average of all values within a receptive field. This is a suitable pooling type for large receptive field sizes, since all elements in the receptive field uniformly contribute to the output.

Global pooling While a CNN of stacked convolutional layers is a powerful hierarchical feature extractor, the output structure is not suited for the most tasks CNNs are used for. Note that the output feature matrix at the end of the convolutional layers is 3-dimensional, while most tasks require flat vectors. A common task is image classification, which requires as output a vector consisting of as many entries as there are classes (as described in Section 2.3.2. There are two ways of restructuring the 3-dimensional feature matrix into such a vector. The first is *global pooling* [Lin et al., 2014], which is a pooling layer with a receptive field size that is equal to the total size of the feature matrix. This intuitively reduces the spatial dimensions of the output feature map (width and height, respectively) to 1. Since the receptive field size is very large, average-pooling is used instead of max-pooling. Just as with local pooling layers, global pooling does not change the non-spatial dimension of the vector. Hence, the channel dimension size of the global pooling output vector is equal to the filter size of the preceding convolutional layer f_{n-1}, which is a hyperparameter that can be freely chosen. In other words, the output of the global pooling layer is of dimensions $1 \times 1 \times f_{n-1}$. Since the spatial dimensions are 1, this matrix can be squeezed into a vector of dimension f_{n-1}. Therefore, by setting the filter size of the ultimate convolutional layer f_{n-1} to the output size required for the specific downstream task l (for image classification, this would be the number of classes) and applying global pooling afterwards, the output of the CNN can be structured in a way that is suitable for downstream tasks. Note, that this requires very little additional parameters. As the global pooling layer is parameter-free and parameters in a convolutional layer are shared across space, the parameters added to the network are equivalent to the parameters of the final convolutional layer $c^2 * f_{n-2} * f_{n-1}$, where $f_{n-1} = l$.

A different way to restructure the 3-dimensional output matrix of convolutional layers

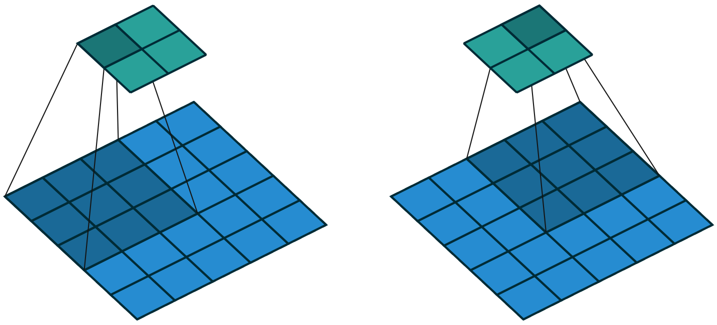


Figure 2.19: Example of strided convolution without padding by Dumoulin and Visin [2016]. The left image depicts the receptive field used to compute the output element at location (1,1) in the output map. The right image depicts the receptive field used to compute the output element at location (2,1).

$\mathbf{o} \in \mathbb{R}^{W \times H \times f_{n-1}}$ is to directly flatten it into a vector $\mathbf{o} \in \mathbb{R}^{W*H*f_{n-1}}$, which is a differentiable operation. This is then followed up by a fully connected hidden layer with a size equivalent to the required output size l. The output of this layer would then be l. While this is conceptionally simpler than the global pooling approach described above, it introduces more parameters. Since the layer is fully connected, this approach requires $W * H * f_{n-1} * l$ parameters, while the global pooling approach only requires $c^2 * f_{n-2} * f_{n-1}$ parameters, where $c \ll W$ and $c \ll H$. Furthermore, since the spatial dimensions are reduced to 1 in a parameter-free way, the global pooling approach is more robust to spatial translations than the fully connected approach.

Strided convolution The convolutional layers described above use the convolution operation defined in Equation (2.17). This convolution is a special case of a strided convolution with a stride of $s = 1$. Intuitively, the receptive field on which the kernel (or, filter) is applied on starts at the beginning of the width dimension and is moved horizontally by successive steps in the width dimension. After it reaches the end of the input matrix in the width dimension, it is moved vertically by one step in the height dimension and back to the beginning in the width dimension. Now, the process repeats until the receptive field has traversed the whole input map.

$$\mathbf{o}_{i,j} = \sum_{k=0}^{c} \sum_{l=0}^{c} \mathbf{w}_{k,l} \mathbf{x}_{(s-1)*i+k,(s-1)*j+l} \tag{2.18}$$

The step size by which the receptive field is moved across the input map is also called *stride*. In strided convolutions, the stride is a hyperparameter that can be freely changed. This operation is defined in Equation (2.18). In Equation (2.18), the stride is denoted by $s \in [1, \min(W, H)]$. Note that setting the stride to $s = 1$ eliminates the stride term, leading to the convolution defined in Equation (2.17). Setting the stride $s > 1$ leads to a

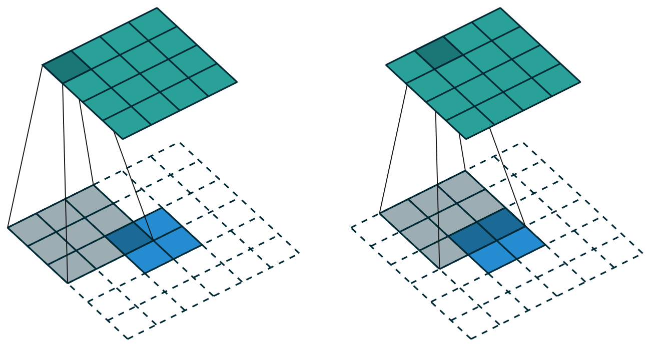


Figure 2.20: Example of a transposed convolution without padding by Dumoulin and Visin [2016]. The left image depicts the receptive field used to compute the output element at location (1,1) in the output map. The right image depicts the receptive field used to compute the output element at location (2,1).

larger spacing between the receptive fields, which effectively reduces the amount of times the kernel is applied in order to produce an entry in the output matrix. Therefore, this leads to an output matrix that is smaller than the input matrix. In other words, strided convolution with $s > 1$ leads to a *downsampling* of the input matrix. In this aspect, it is quite similar to a pooling operation. An example of strided convolution for $s = 2$ is depicted in Figure 2.19.

Transposed Convolution The transposed convolution (also called deconvolution or fractionally strided convolution) operation is the inverse to strided convolution. While strided convolution downsamples the input matrix and therefore reduces the spatial dimensions, transposed convolution upsamples the input matrix in order to increase the spatial dimensions. A visual example of transposed convolution can be seen in Figure 2.20. Intuitively, the input matrix is padded with empty (or, zero-valued) elements in order to produce an output matrix with increased spatial dimensions. Here, the stride hyperparameter can be interpreted as the stride of the output matrix instead of the input matrix.

CoordConv An important property of convolutional layers is that they are invariant to spatial translation. However, for some tasks, this property is actually harmful. A simple example of such a task is mapping carthesian coordinates into one-hot pixel encodings, i.e., to generate an image in which the pixel at the input location is marked. In order to relax translation-invariance, Liu et al. [2018] introduced the CoordConv layer, which is depicted in Figure 2.21. Let $\mathbf{x} \in \mathbb{R}^{W \times H \times C}$ be the input matrix to a convolutional layer. In order to preserve spatial information, explicit coordinate data is concatenated to the non-spatial dimension of the input matrix. In the usual case of 2-dimensional space, the location in space can be represented by two coordinates (i, j), where $i \in [0, W)$ denotes the location in the width dimension and $j \in [0, H)$ denotes the location in the height

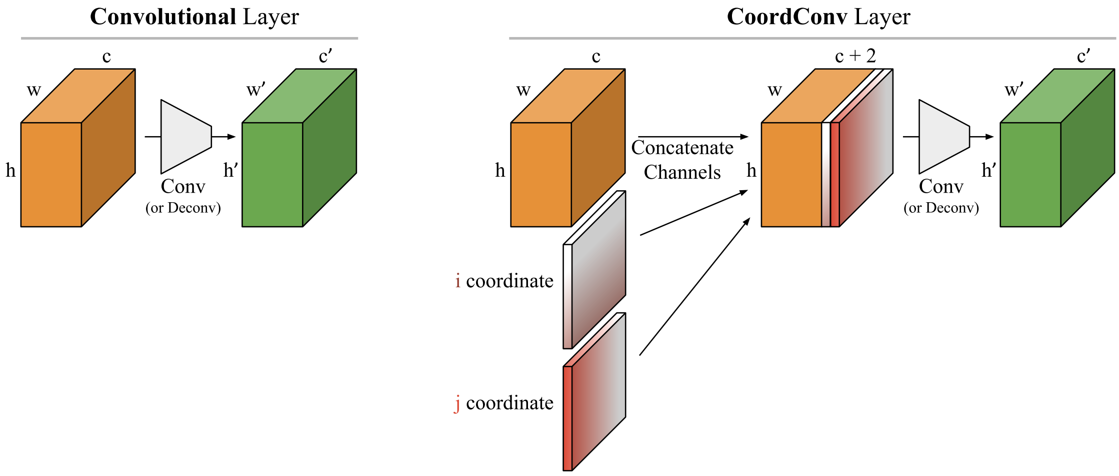


Figure 2.21: Comparison of a conventional convolution layer and a CoordConv layer for input data in 2-dimensional space [Liu et al., 2018].

dimension. For each spatial axis, a matrix $\mathbf{c} \in R^{W \times H}$ is concatenated to the depth dimension of x. In the 2-dimensional case, one coordinate matrix $\mathbf{c}_i$ is successively filled with increasing integers column-wise, while another coordinate matrix $\mathbf{c}_j$ successively filled with increasing integers row-wise. Now, by linearly combining the concatenated coordinate vectors, it is possible to uniquely identify the spatial location of each element in the original input data. In practice, the values in the coordinate vectors $c \in \mathbf{c}$ are scaled to $c \in [-1, 1]$ in order to fit the scale of other input data.

Since CoordConv layers - just like convolutional layers - are fully connected in the non-spatial dimension, the explicit concatenated spatial information is used in the computation of every element in the output matrix. Since the connection weights to the coordinate vectors are freely changeable, the model is able to learn whether to use spatial information depending on the task. Liu et al. [2018] showed that CoordConv layers outperform conventional convolutional layers on a variety of tasks such as image generation, object detection and image classification.

1-dimensional convolution Motivated by a desire to apply CNNs on sequential data such as text and inspired by Time Delay Neural Networks (TDNNs) [Bottou et al., 1989], Collobert et al. [2011] introduce 1-dimensional convolutional layers, which operate on only one spatial dimension. This makes it possible to apply convolutions on sequential (or, temporal) input data, which is often arranged using using 2-dimensional matrices $\mathbf{x} \in \mathbb{R}^{t \times f}$, where t is the sequence length and f is the number of features per element in the sequence. By interpreting t as singular spatial dimension (i.e., assuming linear space along a single axis), it is possible to define 1-dimensional (or, temporal) convolutional layers that perform the same operations as 2-dimensional convolutional layers do on input data in 2-dimensional space.

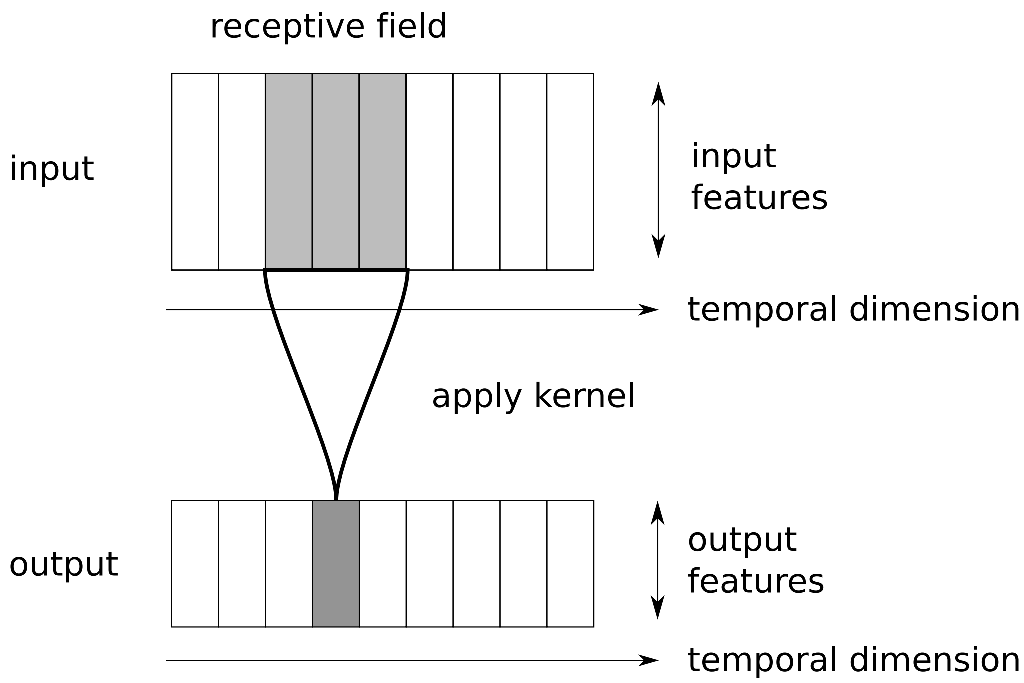


Figure 2.22: Schema of a 1-dimensional convolution.

$$h_{i,j} = \sum_{k=0}^{c} \mathbf{w}_{j,k} \mathbf{x}_{i+k} \tag{2.19}$$

Let $\mathbf{x} \in \mathbb{R}^{t \times f}$ be the input sequence, c denote the receptive field size and $\mathbf{h} \in \mathbb{R}^{t_h \times f_h}$ be the hidden layer output, where just like in 2-dimensional convolutional layers $t_h = t$ assuming padding is used and f_h is the filter size. Then, the 1-dimensional convolutional layer is defined by Equation (2.19), where $h_{i,j}$ is the element at location (i, j) in the output matrix $\mathbf{h}$ and $\mathbf{w} \in \mathbb{R}^{f_h \times c \times f}$ is the kernel, i.e., the parameters or connection weights. The output $\mathbf{h}$ has the same dimensionality as the input sequence, with a singular spatial (or, temporal) dimension and a channel dimension the size of the numbers of filters. This way, the output of this 1-dimensional convolutional layer can in turn be used as input for a successive 1-dimensional convolutional layer, just like 2-dimensional convolutional layers. A visual example can be seen in Figure 2.22.

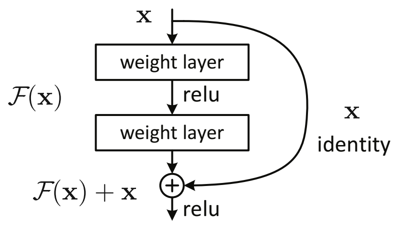


Figure 2.23: Residual learning block [He et al., 2016a].

Architectures

While the above sections introduced the main building blocks of CNNs, this section describes CNN architectures that combine them in a way that has led to empirical successes [Russakovsky et al., 2015].

As described in Section 2.3.1, MLPs generally perform better the deeper they are, i.e., the more hidden layers are used. This is especially true for CNNs, which are explicitly designed to exhibit hierarchical representation learning [Fukushima and Miyake, 1982, Szegedy et al., 2015]. However, it has been shown that CNNs that exceed a certain depth threshold actually underperform [He et al., 2016a]. In order to solve this problem, He et al. [2016a], Srivastava et al. [2015] propose to computationally connect layer outputs that are far away in the graph. This way, the learning signal does not just propagate between directly neighbouring layers, but can *skip* layers. Hence, this connection of layers at different depths is called *skip connection*. Skip connections increase the diversity of paths to deeper layers, resulting in improved accuracy and faster convergence during training.

A simple skip connection proposed by He et al. [2016a] is the residual connection, i.e., to simply sum the outputs of a hidden layer with the outputs of a hidden layer at a different depth. This residual connection is depicted in Figure 2.23. This figure shows two arbitrary hidden layers in a deep CNN, with $\mathbf{x}$ denoting the output of the preceding hidden layer, which is used as input for the first hidden layer. At the end of the block, the output of the second hidden layer is connected through addition with the identity of the output of the hidden layer preceding this block. This addition with the identity function is parameter-free and differentiable. This way, there are now two paths the signal can be propagated to, without increasing the parameter size. One path includes the intermediate first hidden layer, while the other one skips it. Note, that for the matrix addition to work both summands need to have the same dimension.

The ResNet architecture proposed by [He et al., 2016a] stacks a large number of residual blocks in order to form a deep CNN which does not suffer from convergence issues. This CNN architecture is depicted in Figure 2.24. The CNN consists of an initial convolutional layer, followed by a max-pooling layer, 16 residual blocks, a gloval average-pooling layer and an ultimate fully connected layer. In total, this architecture consists of 34 layers. The input image is resized to a width and height of 224 pixels. This is a common practice for CNNs, since high resolution input images would lead to a higher number of parameters. The first convolutional layer performs strided convolution with a stride of $s = 2$, receptive field size of $c = 7$ and a channel size (or, filter size) of $f = 64$. Since $s = 2$, the output feature map of this layer has a width and height of $224/2 = 112$. Afterwards, the max-pooling layer with a receptive field size of 2 further reduces this to 112/56. This output is then used as input for the first of a sequence of 16 residual blocks. The convolutional layers in all residual blocks are performed with a receptive field size of $c = 3$. Intermittently, a strided convolution with $s = 2$ is used in order to halve the output size. The convolutional layer immediately following such a halving has double the filter size of the preceding layer $f_{i+1} = f_i * 2$. All other convolutional layers have the same filter size as their predecessor.

Recall that for addition in the residual connection, the summands need to have the same dimension. This means, given that layers i and its successor j have a different filter size than preceding layer h, a residual connection of h to j is not possible. In order to enable such a connection, a convolutional layer with receptive field size $c = 1$ is applied on the output of the preceding layer $\mathbf{o}_h$ before being added to the output of the connected layer $\mathbf{o}_j$. Crucially, the filter size of this 1x1-convolution is identical to the the filter size of the succeeding layers i and j. Since $c = 1$, the convolution preserves the spatial dimensions of the output, but changes the filter size to match layers i and j. This *projection shortcut* enables residual connections across layers with different filter sizes, but also introduces (a low amount of) additional parameters. It is preferable to keep filter sizes constant in order to use parameter-free residual connections as much as possible. In the few instances in which the filter size changes in the ResNet architecture, projection shortcuts have to be used immediately afterwards. These are indicated by dotted arrows in Figure 2.24.

Since the ResNet architecture was trained and tested for an image classification task, the output feature map has to be structured into a vector with a cardinality of 1000 (which is the number of classes in the ImageNet dataset [Russakovsky et al., 2015]). In order to achieve this, global average-pooling is used in order to reduce the output feature matrix into a vector with cardinality 512, which is the filter size of the ultimate convolutional layer. Afterwards, a fully connected layer is used to produce an output vector of cardinality 1000.

The 34-layer ResNet described above was empirically shown to solve the degradation problems experienced by deep CNN without residual connections [He et al., 2016a]. Therefore, He et al. [2016a] introduced ResNet architectures that were even deeper. For computational efficiency, residual blocks are replaced by *bottleneck* blocks, which are depicted in Figure 2.25. Here, the two convolutional layers of residual blocks are replaced with three convolutional layers. The first convolutional layer has a receptive field size $c = 1$ and a filter size $f = 64$. This layer solely serves to reduce the filter size. The succeeding convolutional layer with a receptive field size of 3 performs the usual feature extraction on this reduced filter size. The final convolutional layer again has a receptive field size of 1 and a filter size of 256. As an inverse to the first layer, it serves to increase the filter size to the same size it is in output of the preceding block. This way,

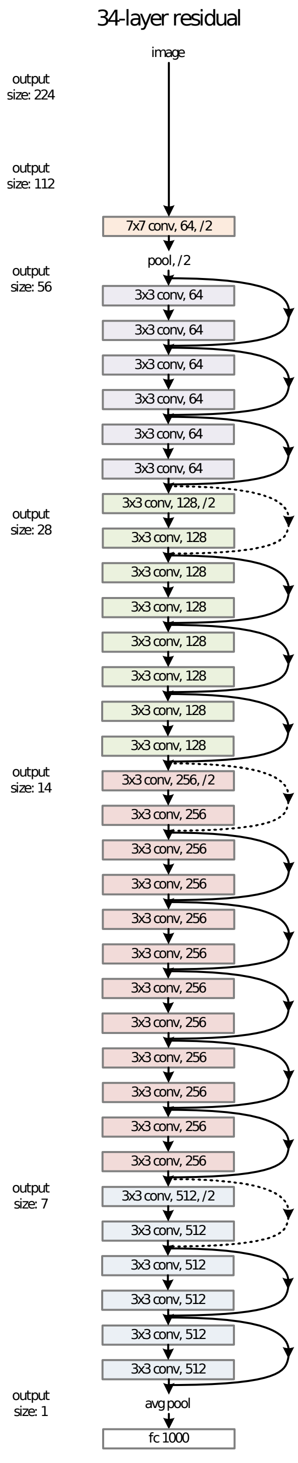


Figure 2.24: ResNet architecture design with 34 layers and residual blocks instead of bottleneck blocks [He et al., 2016a].

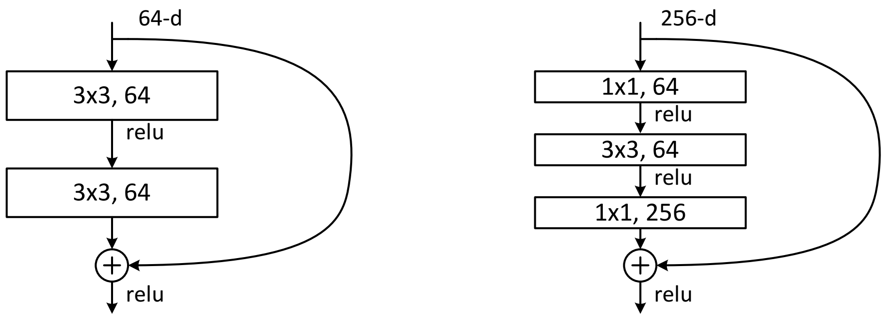


Figure 2.25: Comparison of a residual block (left) and a bottleneck block (right) [He et al., 2016a].

the convolutional layer performing feature extraction is a bottleneck with smaller input and output filter size. This serves both as regularization and to increase computational efficiency.

By replacing all residual blocks in the ResNet-34 architecture with bottleneck blocks, the number of layers is increased to 50. He et al. [2016a] show that this model already performs better than ResNet-34. He et al. [2016a] build even deeper models by stacking more bottleneck blocks, arriving at a ResNet model with 152 layers, which yielded the best empirical results.

The ResNet architecture has been widely used to create deep CNN architectures. There exists a range of extensions and modifications to the original architecture, including ResNetv2 [He et al., 2016b], Wide ResNet [Zagoruyko and Komodakis, 2016], ResNeXt [Xie et al., 2017] and most recently ConvNeXt [Liu et al., 2022].

2.3.4 Recurrent Neural Networks

While the MLP is a general purpose architecture and the CNN has an inductive bias for spatial data, the recurrent neural network is a neural network architecture with an inductive bias for sequential (or, temporal) data. The original architecture introduced by Rumelhart et al. [1986] equips neural networks with memory capabilities. In contrast to other neural network architectures such as MLPs or CNNs, individual forward passes on inputs are not independent. Let $\mathbf{x}_n$ be an input sequence. Just like with neural networks, recurrent neural networks perform a forward pass according to $f(\mathbf{x}_0; \boldsymbol{\theta})$. Contrary to neural networks, the forward pass for the next input $\mathbf{x}_1$ depends on the forward pass of the previous input, i.e., $f(\mathbf{x}_1; \boldsymbol{\theta}; f(\mathbf{x}_0; \boldsymbol{\theta}))$. This loop-like functionality enables RNNs to take previous inputs and predictions into account when computing new predictions. The simple RNN architecture is defined in Equations (2.20) and (2.21).

$$\mathbf{h_t} = \phi(\mathbf{h}_{t-1}\boldsymbol{\theta}_h + \mathbf{x}_t\boldsymbol{\theta}_x) \tag{2.20}$$

$$\mathbf{o}_t = \mathbf{h_t}\boldsymbol{\theta}_o \tag{2.21}$$

Note that all variables in Equations (2.20) and (2.21) are temporal, i.e., depend on t. Only the learnable parameters $\boldsymbol{\theta}$ are constant. $\mathbf{h}_t$ is the state of recurrent neural network. It represents all individual inputs and state changes up to $\mathbf{x}_t$. The output $\mathbf{o}_t$ is computed by taking the current state $\mathbf{h}_t$ into account. Thus, a RNN can natively handle temporal inputs.

Recurrent neural networks are optimized in the same way as standard neural networks. However, due to the temporal component introduced in Equations (2.20) and (2.21), the gradient can not be computed using standard backpropagation. The gradients needed for loss function optimization are computed using the backpropagation-through-time algorithm [Robinson and Fallside, 1987].

In practice, standard recurrent neural networks are not able to generalize well and do not take long-term dependencies into account because they suffer from vanishing or exploding gradients [Hochreiter, 1991, Bengio et al., 1994].

Gated Recurrent Units

In order to alleviate the convergence problems of standard recurrent neural networks, Hochreiter and Schmidhuber [1997] introduced the Long Short-Term Memory (LSTM) recurrent neural network. In addition to the hidden state $\mathbf{h}_t$, it introduces a cell state $\mathbf{c}$ and a corresponding set of update equations which do not suffer from gradient problems. A simplified variant of the LSTM architecture are Gated Recurrent Units (GRUs) [Cho et al., 2014a]. Its authors claim that GRUs work better than LSTMs. Equations (2.22) to (2.25) detail the update procedure similar to Equations (2.20) and (2.21). Note that $\mathbf{o}_t = \mathbf{h}_t$.

$$\mathbf{z}_t = \phi_{\text{sigmoid}}([\mathbf{h}_{t-1}, \mathbf{x}]\boldsymbol{\theta}_z) \tag{2.22}$$

$$\mathbf{r}_t = \phi_{\text{sigmoid}}([\mathbf{h}_{t-1}, \mathbf{x}]\boldsymbol{\theta}_r) \tag{2.23}$$

$$\tilde{\mathbf{h}}_t = \phi_{\tanh}([\mathbf{r}_t\mathbf{h}_{t-1}, \mathbf{x}]\boldsymbol{\theta}_h) \tag{2.24}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t)\mathbf{h}_{t-1} + \mathbf{z}_t\tilde{\mathbf{h}}_t \tag{2.25}$$

It is still unknown whether GRUs truly improve upon LSTMs. A large-scale study [Greff et al., 2017] compared LSTM variants on speech recognition, handwriting recognition and music modeling tasks and found no significant difference. Another study [Jozefowicz et al., 2015] used a different hyperparameter optimization algorithm and compared LSTMs and GRUs on arithmetic, XML modeling, language modeling and music modeling tasks. They conclude that GRUs perform better than LSTMs.

CNN or RNN?

As discussed in Section 2.3.3 and Section 2.3.4, both 1-d convolutional and recurrent neural networks can handle temporal data. While only RNNs can properly handle sequences of unknown length, for input data of fixed length, it is of interest to know which architecture is better suited. This comparison was investigated by Yin et al. [2017], Bai et al. [2018]. Both come to the conclusion that 1-dimensional CNNs lead to better and faster convergence than RNNs on sequence classification tasks. Keep in mind, however, that the empirical results using toy problems might not generalize.

2.3.5 Transformer

The Transformer introduced by Vaswani et al. [2017] is a neural network architecture that essentially combines the advantages of RNNs and CNNs. Like RNNs, it can be used for sequential data, but like CNNs it can be efficiently computed on GPUs using matrix multiplications. The Transformer consists almost solely of *self-attention* modules, which are explained below.

Self-Attention

Attention is a general purpose mechanism for neural networks introduced by Bahdanau et al. [2015] and was originally intended for machine translation RNNs. It allows to assign each element $\mathbf{x}_i$ of an input set $\mathbf{X}$ an attention weight α_i. Crucially, this mechanism is differentiable, so it can be fitted along with the neural network model.

Originally, attention was intended to compute values using both an input sequence (text in Bahdanau et al. [2015]) and an output sequence (RNN hidden states in Bahdanau et al. [2015]). However, the form of attention used for Transformers operates over a single input sequence only. This form is called *self-attention* (or inter-attention).

Let $\mathbf{X} \in \mathbb{R}^{n \times m}$ be a set of n elements with m features. In the general form, the self-attention mechanism transforms $\mathbf{X}$ into a *contextualized* set $\mathbf{Y} \in \mathbb{R}^{n \times m}$. The resulting set is of the same length as $\mathbf{X}$ and $\mathbf{y}_i \in \mathbf{Y}$ can be interpreted as representing the value of $\mathbf{x}_i \in \mathbf{X}$ taking into consideration all other elements of $\mathbf{X}$. Mathematically, this can be achieved by first computing *attention weights* $\mathbf{A}$ as the dot-product of $\mathbf{X}$ with itself, as can be seen in Equation (2.26). This equation uses the unit softmax function defined in Equation (2.27) to produce a pseudo-probability distribution from the unbounded attention scores, i.e., to bound them to $[0, 1]$ and let them sum to 1. The contextualized set $\mathbf{Y}$ can then be computed as the weighted sum of $\mathbf{X}$ using the attention weights, as defined in Equation (2.28).

$$\mathbf{A} = \text{softmax}(\mathbf{X}\mathbf{X}') \tag{2.26}$$

$$\mathrm{softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{|\mathbf{x}|}\exp(x_j)} \tag{2.27}$$

$$\mathbf{y}_i = \sum_{j=1}^{n} \alpha_{ij}\mathbf{x}_j \tag{2.28}$$

In conclusion, the element $\mathbf{y}_i \in \mathbf{Y}$ is computed by *attending* to individual elements of $\mathbf{X}$ with different weights. This attention variant using a weighted sum introduced here is also called *soft attention*, since all input elements contribute to the output. There also exists hard attention described in Xu et al. [2015], which explicitly selects elements instead of using a weighted sum of all elements. Since this is not differentiable, it is more complex to integrate it in the optimization procedure.

This self-attention mechanism can be further extended with learnable parameters. Notice in Equations (2.26) and (2.28) that $\mathbf{X}$ is used three times in the self-attention mechanism. Different learnable parameters $\mathbf{W} \in \mathbb{R}^{m\times m}$ are used for these three occurrences. To distinguish them more easily, these are defined analogous to a continuous dictionary as query $\mathbf{Q} = \mathbf{X}\mathbf{W}_\mathbf{Q}$, key $\mathbf{K} = \mathbf{X}\mathbf{W}_\mathbf{K}$ and value $\mathbf{V} = \mathbf{X}\mathbf{W}_\mathbf{V}$. In Equation (2.29), the query and the key are matched to produce a compatibility (i.e., the attention weights), which is then used in Equation (2.30) to retrieve the values. It is also clear that this attention variant is called *self-attention* because $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ are computed using the same input $\mathbf{X}$.

$$\mathbf{A} = \mathrm{softmax}(\mathbf{Q}\mathbf{K}') \tag{2.29}$$

$$\mathbf{Y} = \mathbf{A}\mathbf{V} \tag{2.30}$$

The self-attention variant described until now is called *dot-product attention* [Luong et al., 2015, Graves et al., 2014]. There also exists *additive attention* [Bahdanau et al., 2015]. This approach first computes attention scores s_{ij} by concatenating $\mathbf{q_i}$ and $\mathbf{k_j'}$ and using the concatenated vector as an input to a single layer MLP. This can be seen in Equation 2.31. In this equation, $\mathbf{q}_i \| \mathbf{k}_j \in \mathbb{R}^{nm}$, $\mathbf{W_A} \in \mathbb{R}^{nm}$ are the learnable parameters and ϕ is a non-linearity such as the tanh function. Then in Equation (2.32) the softmax function is used to compute the attention weights.

$$s_{ij} = \phi(\mathbf{W_A}[\mathbf{q}_i \| \mathbf{k}_j]) \tag{2.31}$$

$$\alpha_{ij} = \mathrm{softmax}(\mathbf{s})_{ij} \tag{2.32}$$

In practice, additive attention leads to better results than dot-product attention if the feature dimension m is large [Britz et al., 2017]. However, it can be seen that, to compute the entire attention matrix $\mathbf{A}$, additive attention needs to use the neural network nm times. This cannot be restructured into a single matrix multiplication due to the non-linearity. Contrary to that, dot-product attention can simply be computed using a matrix multiplication, which is more suited for GPU workloads. To remedy this situation, *scaled* dot-product attention [Vaswani et al., 2017] has been introduced. This simply normalizes the dot product by $1/\sqrt{m}$ to counteract the adverse effect of large values of m and has largely replaced additive attention.

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{QK}'}{\sqrt{m}}\right) \tag{2.33}$$

Another way of stabilizing self-attention is multi-head attention [Vaswani et al., 2017], which is analogous to using multiple channels in CNNs. Multiple heads of the self-attention function h are computed in parallel using different learnable parameters in Equation 2.34. The h output matrices are concatenated and projected to the original dimension $\mathbb{R}^{n\times m}$ using the learnable linear transformation $\mathbf{W_Y} \in \mathbb{R}^{n\times nh}$ in Equation 2.35. Intuitively, this allows to attend to different embeddings simultaneously. Vaswani et al. [2017] set $h = 8$ for their architecture.

$$\mathbf{Y}^{\text{head}_i} = \text{Self-Attention}(\mathbf{X}; \mathbf{W}^i_\mathbf{Q}, \mathbf{W}^i_\mathbf{K}, \mathbf{W}^i_\mathbf{V}). \tag{2.34}$$

$$\mathbf{Y} = \mathbf{W_Y}\left(\|_{i=0}^{h} \mathbf{Y}^{\text{head}_i}\right) \tag{2.35}$$

The main computational problem of standard self-attention is that its space and time complexity is quadratic in the set length n. This quickly becomes a problem for larger values of n. Multiple approaches were proposed to modify self-attention to achieve lower runtime, memory or IO complexity [Choromanski et al., 2021, Dao et al., 2022].

Another approach to solve this computational problem is local attention [Luong et al., 2015]. The self-attention variant described above can be seen as *global* attention, since it uses the entire input set $\mathbf{X}$ to compute each output vector $\mathbf{y}_i$. In local attention, only a constant number of neighbours is used. This is computationally more efficient, but restricts the contextualized output vectors from taking longer-range dependencies into account. In practice, most works utilize standard self-attention and deal with the complexity by increasing hardware resources.

Standard Transformer

While (self-)attention was originally only used as an add-on for RNNs, the Transformer architecture introduced by Vaswani et al. [2017] consists solely of global scaled self-attention modules stacked with an MLP. They showed that this is not only simpler but actually preferable to using RNNs. While the Transformer uses a fixed number of self-attention modules, an RNN requires n sequential operations. This is an advantage,

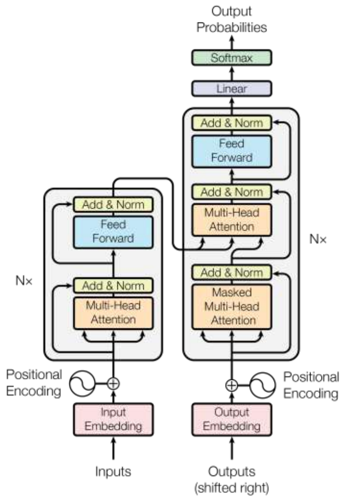


Figure 2.26: Transformer architecture by Vaswani et al. [2017].

since it has been established that a shorter path length between two positions makes it easier to learn long-range dependencies [Kolen and Kremer, 2001].

The Transformer architecture is depicted in Figure 2.26 and consists of stacked transformer blocks, with each block consisting of multi-head self-attention and layer normalization [Ba et al., 2016] (similar to batch normalization as described in Section 2.3.1) followed by a simple MLP and layer normalization. The MLP is position-wise, i.e., it is applied to each output vector $\mathbf{y}_i$ individually. The architecture employs residual connections [He et al., 2016] around each individual block step.

The Transformer architecture can be used for any kind of real-valued set data. Originally, it was intended for Natural Language Processing (NLP) tasks and thus to beused with text data as input. However, text data is sequential in nature, while the Transformer architecture does not preserve order information due to the permutation-equivariant self-attention layers. This is similar to using 1-dimensional CoordConv layers as described in Section 2.3.3. In order to be able to use sequential data, the Transformer architecture concatenates a positional encoding to each input element before applying self attention. The Transformer architecture employs global attention and thus computes the attention of all input pairs.

Since the Transformer continually leads to new successes in NLP [Devlin et al., 2019, Brown et al., 2020, Touvron et al., 2023], there have been numerous attempts to apply it on visual data. These Vision Transformers (ViTs) perform comparably and in some cases outperform CNNs on a variety of tasks such as image classification and object detection [Dosovitskiy et al., 2021, Liu et al., 2021].

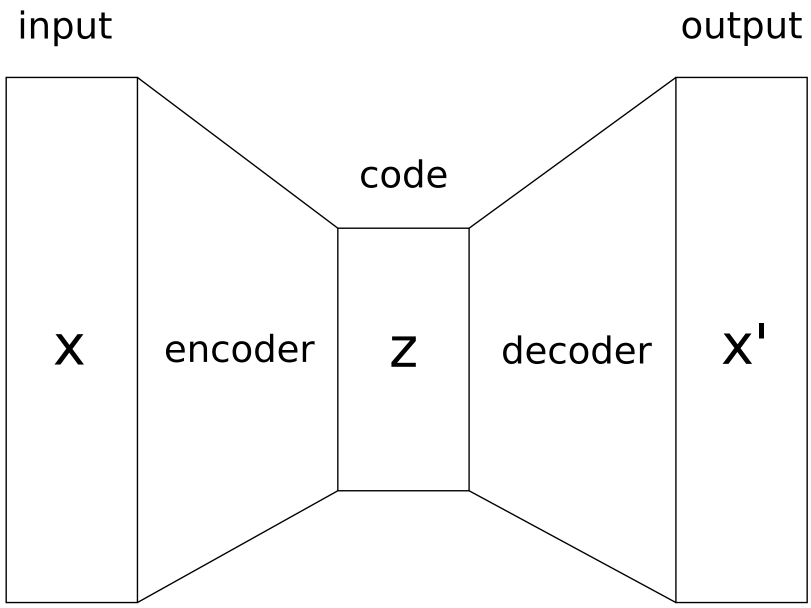


Figure 2.27: Schema of the autoencoder architecture.

2.3.6 Encoder-Decoder Framework

An important general framework for generative deep learning architectures is the encoder-decoder framework. To motivate this, first, the *autoencoder* architecture [Kramer, 1991] is introduced.

Recall that neural networks are universal function approximators, i.e., they approximate an arbitrary real-valued ground truth function $F(\mathbf{x}) = \mathbf{y}, F : \mathcal{X} \to \mathcal{Y}$ using freely changeable parameters $\boldsymbol{\theta}$ as $f(\mathbf{x}; \boldsymbol{\theta})$. In many tasks, the input and output data are part of different vector spaces, e.g. in image classification $\mathbf{x} \in \mathcal{X}$ being an image and $\mathbf{y} \in \mathcal{Y}$ being the class probabilities. The autoencoder architecture was introduced by Kramer [1991] to solve the task of condensing the input $\mathbf{x}$ into a an efficient (i.e., smaller) code $\mathbf{z} \in \mathcal{Z}$, which can be used to fully reconstruct $\mathbf{x}$. It must hold, then, that this code contains all essential information about the input data. This code is referred to as *latent code*, or, since it is represented as a vector, *latent vector*.

For the purpose of finding efficient encodings, the autoencoder consists of two neural networks. The *encoder* e defined in Equation (2.36) is an MLP which maps the input $\mathbf{x} \in \mathcal{X}$ to the latent vector $\mathbf{z} \in \mathcal{Z}$ using freely changeable parameters $\boldsymbol{\theta}_e$. In turn, the *decoder* d defined in Equation (2.37) is an MLP which maps the latent vector $\mathbf{z}$ to an output vector $\mathbf{x}' \in \mathcal{X}$ with $\mathbf{x}' \approx \mathbf{x}$ (or, in the optimal case $\mathbf{x}' = \mathbf{x}$) using freely changeable parameters $\boldsymbol{\theta}_d$. The complete autoencoder is then defined by Equation (2.38) and depicted in Figure 2.27. The autoencoder is optimized using gradient descent of a loss function measuring the distance between $\mathbf{x}'$ and $\mathbf{x}$ such as the L^1 or L^2 distance described in Section 2.3.2. Furthermore, an important design consideration is that the latent vector is smaller than the input vector, since if $\dim(\mathbf{z}) = \dim(\mathbf{x})$, the autoencoder might converge to the identity function as local optimum.

$$\mathbf{z} = e(\mathbf{x}; \boldsymbol{\theta}_e) \tag{2.36}$$

$$\mathbf{x}' = d(\mathbf{z}; \boldsymbol{\theta}_d) \tag{2.37}$$

$$\mathbf{x}' = d(\mathbf{z}; \boldsymbol{\theta}_d) = d(e(\mathbf{x}; \boldsymbol{\theta}_e); \boldsymbol{\theta}_d) \tag{2.38}$$

Autoencoders were proposed for dimensionality reduction, since the latent vector theoretically contains all necessary information of the input data in smaller form. This can be useful as a compression technique. Additionally, it is possible to apply vector space operations on the latent vector that might not have been possible or practical on the input data, such as linear transformations or distance measures. Furthermore, it is possible to use autoencoders for generative tasks by using the decoder of an optimized autoencoder on its own. By sampling an arbitrary $\mathbf{z}_k \in \mathcal{Z}$ and inputting it into an optimized d, it is possible to generate new output data $\mathbf{x}'_k \in \mathcal{X}$ that follows the same distribution of the training input data (e.g. if $\mathcal{X}$ is an image space, generating new images that are similar to the images in the training dataset). In turn, since the encoder is differentiable, optimized encoders can also be used on their own for downstream tasks. For this reason, autoencoders have found a wide range of applications [Cho et al., 2014b, Theis et al., 2017, Hinton and Salakhutdinov, 2006].

By generalizing the autoencoder to inputs and outputs of different distributions, one arrives at the encoder-decoder framework [Sutskever et al., 2014, Mikolov, 2012] as depicted in Figure 2.28, which is used by a wide variety of recently successful neural network architectures such as the Transformer (as explained in Section 2.3.5). As an example, Mikolov et al. [2013] use an encoder-decoder architecture to encode words (i.e., character sequences) into semantically meaningful latent vectors. The encoder-decoder framework is defined in Equation (2.39), which is a generalization of Equation (2.38). Note that the output vector $\mathbf{y} \in \mathcal{Y}$ of the decoder d follows a different distribution than the input vector $\mathbf{x} \in \mathcal{X}$. Furthermore, the optimization objective is not $\mathbf{y} = \mathbf{x}$, as is the case in autoencoders, but is replaced with a task-dependent objective.

$$\mathbf{y} = d(\mathbf{z}; \boldsymbol{\theta}_d) = d(e(\mathbf{x}; \boldsymbol{\theta}_e); \boldsymbol{\theta}_e) \tag{2.39}$$

As an aside, there exist different versions of and modifications to the autoencoder architecture. Chief among them is the variational autoencoder (VAE), which attempts to normalize the latent space $\mathcal{Z}$ s.t. $\forall z_i \in \mathbf{z} \in \mathcal{Z}$, $z_i \sim \mathcal{N}(0, 1)$, where $N(\mu, \sigma)$ is a normal distribution with mean μ and standard deviation σ. In order to achieve this, a loss term which calculates the divergence between $\mathbf{z}$ and the standard-normal distribution $\mathcal{N}(0, 1)$ is added to the autoencoder loss. Usually, Kullback–Leibler (KL) divergence [Kullback and Leibler, 1951] is chosen as loss function.

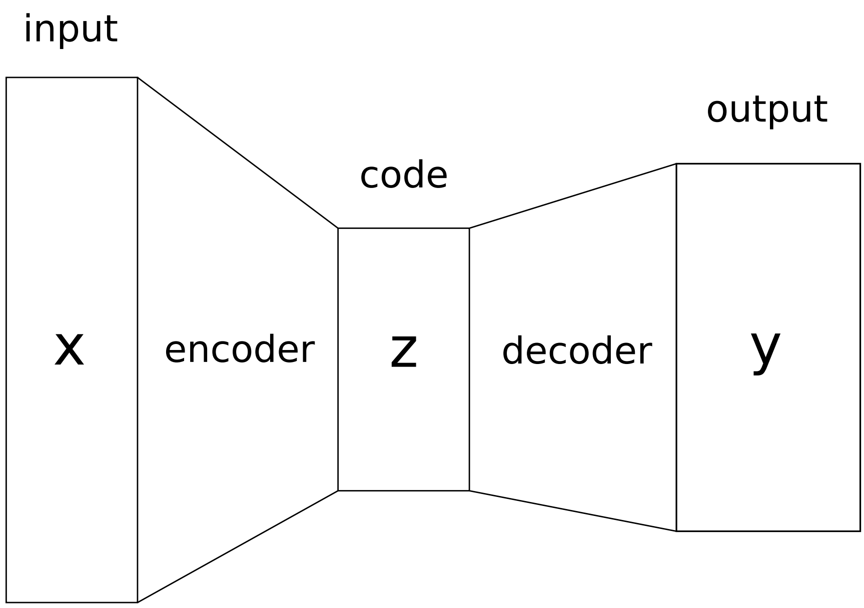


Figure 2.28: Schema of an architecture following the encoder-decoder framework.

Feature Combination

There exist encoder-decoder architectures with multiple encoders but only one decoder, most commonly if input from different modalities is used to generate a single output. In this case, multiple latent vectors have to be combined into a single latent vector suitable for consumption by the decoder. This is referred to as *feature combination*, or *feature fusion*. Two common types of feature combination are introduced in this section.

$$\mathbf{z_o} = \mathbf{z}_0|\mathbf{z}_1|\ldots|\mathbf{z}_N = |_{i=0}^{N}\mathbf{z}_i \tag{2.40}$$

A simple feature combination technique is to concatenate the latent vectors into a single, latent vector, as defined in Equation (2.40). This is parameter-free and quick to compute. Let $|\mathbf{z}_i|$ be the length of an input latent vector. Then the length of the output latent vector is $|\mathbf{z}_o| = \sum_i^N |\mathbf{z}_i|$. Hence, the combined latent vector is of a different length than the input vectors and quickly grows very long. If the decoder requires the latent vector to be of a specific length, or if the concatenated latent vector would be too long, *learned* feature combination is preferable. In this technique, the concatenated latent vectors are input followed by a fully connected layer, which produces a combined latent vector of the hidden layer size, which can be set as a hyperparameter. It is defined by Equation (2.41), where f is a fully connected hidden layer. Note, however, that depending on the length this introduces a considerable amount of parameters. As an aside, learned feature combination by concatenation is similar to additive attention as introduced in Equation (2.31).

$$\mathbf{z_o} = f(|_{i=0}^{N}\mathbf{z}_i; \boldsymbol{\theta}) \tag{2.41}$$

Note that there exists a wide range of feature combination types other than the ones explained above. As an example, the skip connections introduced in Section 2.3.3 are a kind of feature combination by addition.

CHAPTER 3

Related Work

This chapter details existing work on vectorization and vector conversion across different image domains, specifically for the case of line art. Section 3.1 details works that attempt to generate a corresponding vector image given a raster image and is related to the implementation of the deep learning model for line-art vectorization as described in Section 1.4 (see RO1). Section 3.2 explores cross-domain line-art vectorization, which is required for the potential extension of the model into final animation frame to clean animation frame vector conversion. While cross-domain vectorization is not the focus of this work, the goal is to design the model in a way that makes it easily adaptable for this task.

3.1 Line-art Vectorization

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 [Selinger, 2003, Weber, 2002, Noris et al., 2013, Bessmeltsev and Solomon, 2019, Zhang et al., 2022], they do not produce the intended output for this task. As mentioned in Chapter 1, the resulting vector primitives rarely resemble the primitives an artist would draw naturally. Crucially, these algorithms are not differentiable, meaning that they can not be finetuned to vectorize input images across domains. 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. Finally, and somewhat counter-intuitively, a learned method could potentially be faster, since the number of primitives in an animation line-art image is large and traditional methods have a high runtime complexity in the number of primitives. However, note that this

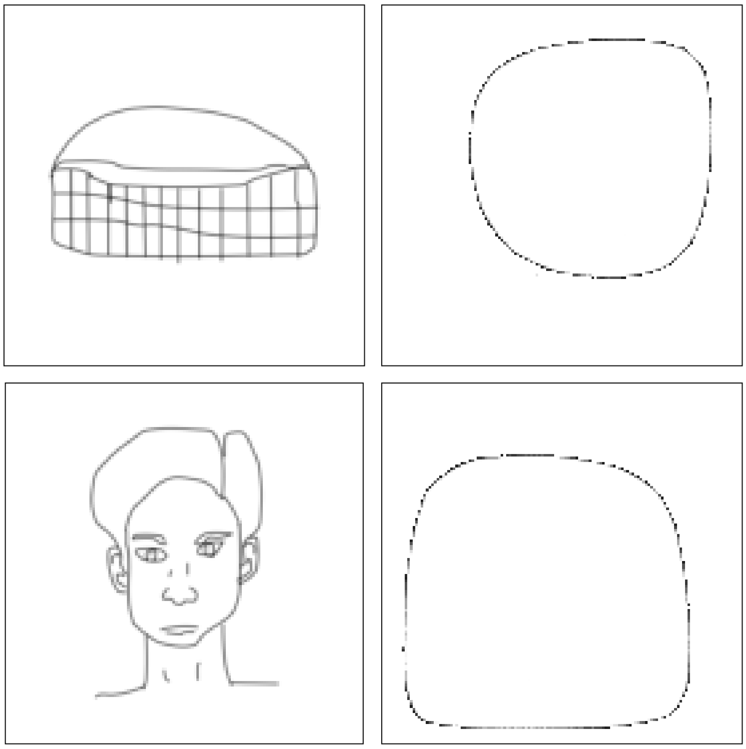


Figure 3.1: Results of Im2Vec [Reddy, 2021] on two simple line-art sketches from Eitz et al. [2012]. The image size is 128x128px.

only applies to a zero-shot model and not to the iterative deep learning models explored in this work.

While image vectorization is not yet a solved task, there have been some recent advances in deep learning for vector images. Reddy [2021] introduce Im2Vec, an encoder-decoder architecture consisting of a CNN encoder and a 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 bezier curves. It can be trained to vectorize raster images without vector supervision (i.e., using only raster images in the ground truth training set). 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 [Li et al., 2020]. 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. Since clean animation line art only includes a subset of possible vector image graphic primitives, this might be avoidable by imposing (heuristic) geometric constraints.

However, our experiments showed that Im2Vec only works on a specific type of image, such as emojis or icons. It does not perform well when trained on line-art images, as can be seen in Figure 3.1. Additionally, there were no experiments in the paper to output more than 4 shapes. It is doubtful whether it is possible to train the RNN decoder to output the large number of bezier curves required for a clean animation frame.

The virtual sketching framework introduced by Mo et al. [2021] 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 bezier curves. Also, it is an iterative model, i.e., the curves are drawn one at a time. 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. Other differences to Im2Vec include using a different differentiable rasterizer [Huang et al., 2019] and a finetuned perceptual loss [Johnson et al., 2016] instead of an L^2 loss. However, since the iterative model is trained mainly by computing a perceptual loss of the whole output image with the input image, the results are not semantically meaningful vector images. So while the model produces a collection of bezier curves that visually resembles the input image at a certain resolution, the vector image does not preserve the topology or meaningful structure which is necessary for clean animation frames. Related work includes Su et al. [2021], who use reinforcement learning as a framework for learning an iterative model in the context of comic line-art vectorization. The same constraints as with the virtual sketching framework apply here. Additionally, comic book line art does not translate well to limited animation production line art, although it would seem so at first glance.

A different approach is to incorporate parts of traditional optimization-based methods. The state-of-the-art of traditional methods was introduced by Bessmeltsev and Solomon [2019]. They attempt to detect X and T-junctions by tracing black pixel orientations with a frame field. However, additionally to the general drawbacks of traditional methods, the resulting method is not robust to more complex junctions with sharp turns, fine details or noise in general. Puhachov et al. [2021] try to improve upon that by using 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. Their method produces good results when applied to clean animation line art. However, resulting vector images contain overparameterized primitives and fail to vectorize more detailed and smaller shapes.

Similarly, other successful methods focus on extracting keypoints, but using a fully learned architecture. Guo et al. [2019] use a multi-task CNN architecture to produce a centerline image and a junction image. Using this information, another CNN extracts the curve topology. The topology image of each curve is then traced using cubic bezier least square fitting. The model is trained using raster supervision and on a synthetically created dataset. Therefore, it does not generalize well on complex real line art. Furthermore, the resulting vector image is constrained by the quality produced by the curve fitting algorithm.

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. Wang and Lian [2021] use both raster and vector supervision to learn a model that generates fonts glyphs given a reference glyph. Since the number of curves required for a glyph is small (≈ 10), the model is not trained iteratively but directly outputs a sequence of drawing commands. Their method is potentially useful, but it is doubtful whether it generalizes to a large number of curves. Gao et al. [2019] use solely vector supervision to reconstruct splines (which are generalizations of bezier curves). However, their approach using a hierarchical RNN is only trained with up to three splines (with 4 to 6 control points). Bhunia et al. [2021] use line vectorization as a self-supervised pretraining task to learn suitable sketch embeddings for downstream tasks. The line vectorization itself is similar to Reddy [2021] in that it uses a CNN as encoder and a RNN as an encoder to generate the whole image at once. Contrary to Reddy [2021] it is trained using vector supervision and constrained to output curves as a sequence of draw commands. Similar to Reddy [2021], the model is only tested with vector images containing a small number of curves.

In a similar vein, a method to generate technical drawings by Egiazarian et al. [2020] is also framed as a line vectorization problem trained solely using vector supervision. It uses the Transformer architecture and is constricted to only handle 10 curves per image. To handle 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. The authors also show that the pure primitive predictions by the Transformer model are lacking, requiring the physics-inspired refinement algorithm, which relies on strong assumptions of the input image. This is displayed in Figure 3.2. Additionally, the model was only tested for two vector primitives: lines and quadratic bezier curves. When applied to clean animation frames, it produces images that are both visually and structurally pretty close to the original at certain parts. However, like Puhachov et al. [2021] it skips certain smaller and more detailed shapes. Quite paradoxically, it also produces a lot of superfluous small curves at some parts.

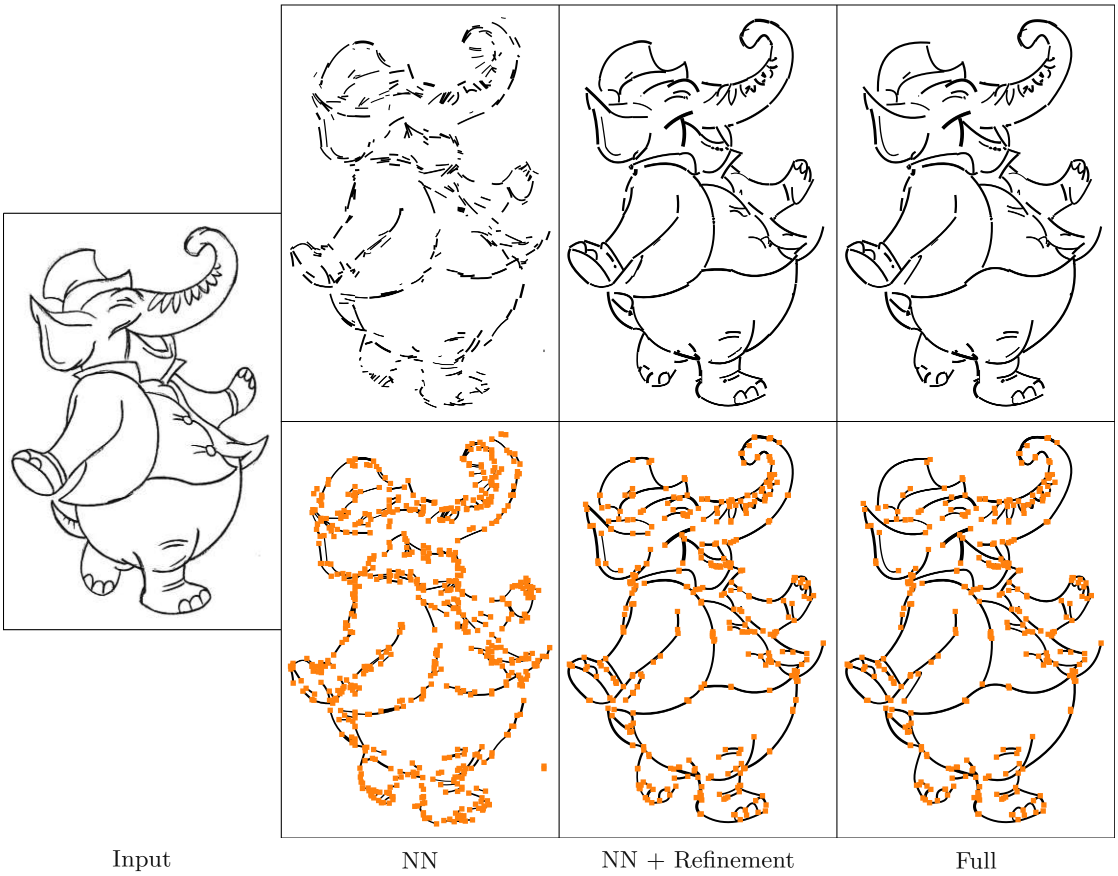


Figure 3.2: Result and intermediates step of the model by Egiazarian et al. [2020] on line art by Huska [2009].

3.2 Cross-Domain Line-Art Vectorization

To our knowledge, final animation frame to clean animation frame conversion has not yet been attempted. This task is related to works attempting to generate vector line art using input images of another domain, such as photos or illustrations.

Mo et al. [2021] provide experiments with generating vector line art using photographs as input. However, the authors concede that the model does not generalize well to complex images and produces artifacts.

The model proposed by Puhachov et al. [2021] can be regarded as the state-of-the-art for sketch to clean vector line image conversion. However, the method relies on strong assumptions regarding the input image, specifically regarding the background-foreground threshold, which prevents the method to be used for drawings with a very noisy background (such as final animation frames).

Li et al. [2019] implement a GAN to generate raster contour sketch images given realistic photos as input. It is doubtful whether the results produced by this method would lead to a semantically meaningful vector image. Either way, Mo et al. [2021] seem to outperform this method (and produce cleaner results due to the fact that the output is restricted to vector primitives).

There exists related work specifically related to anime-style illustrations. [Zhang, 2017] devise a model which produces a raster line art given an illustration. The results contain substantial artifacts and are therefore not quite usable as clean animation line art. The two main problems are lack of high quality training data and pixel-level supervision. The colored-sketch pair dataset normally used for such models [Li, 2017] only superficially resembles clean animation frames, primarily since the line-art images have variable stroke widths.

Zhang et al. [2021] generate manga-style images from illustrations. The generation is constrained by the actual manga creation workflow. The first step of this workflow is the generation of line art given the illustration using a U-Net architecture [Ronneberger et al., 2015]. This could be used to generate a raster line art given a final animation frame. Then, the raster image could be vectorized using the line-art vectorization model. Unfortunately, the data used to train this model is not public. Furthermore, the provided results contain artifacts which will likely make it challenging to correctly vectorize the image.

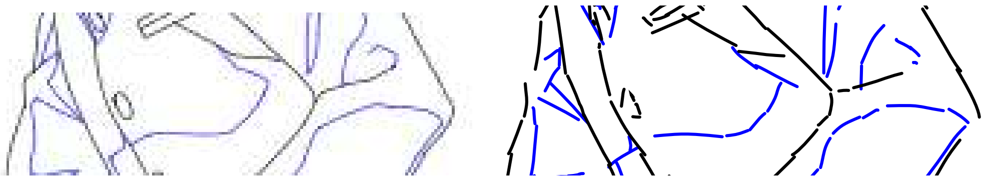
CHAPTER 4

Animation Line-art Vectorization

This chapter describes our work, which attempts to answer the research question posed in Section 1.4, i.e., to what extent it is possible to automatically vectorize clean animation frame line art in a manner that is semantically meaningful. In order to answer this question, a method to automatically vectorize clean animation frame line art is developed based on previous works, which is described in Section 4.1 and depicted in Figure 4.1. This method is trained on a dataset detailed in Section 4.2. We then evaluate, both

(a) The clean animation frame in raster format as input.

(b) The output vector image of the line-art image vectorization method.



(c) Region of the input image.

(d) Region of the predicted vectorization.

Figure 4.1: Sample output vector image of the developed line-art image vectorization method based on the raster clean animation frame provided by Tonari Animation as input. Zooming into the image reveals structural differences.

qualitatively and quantitatively, the extent to which this method and comparable works can automatically vectorize clean animation frame line art on this dataset in Section 4.3. Finally, in Section 4.4, we describe alternative model architectures explored and provide an ablation study evaluating different configurations of the method.

4.1 Method

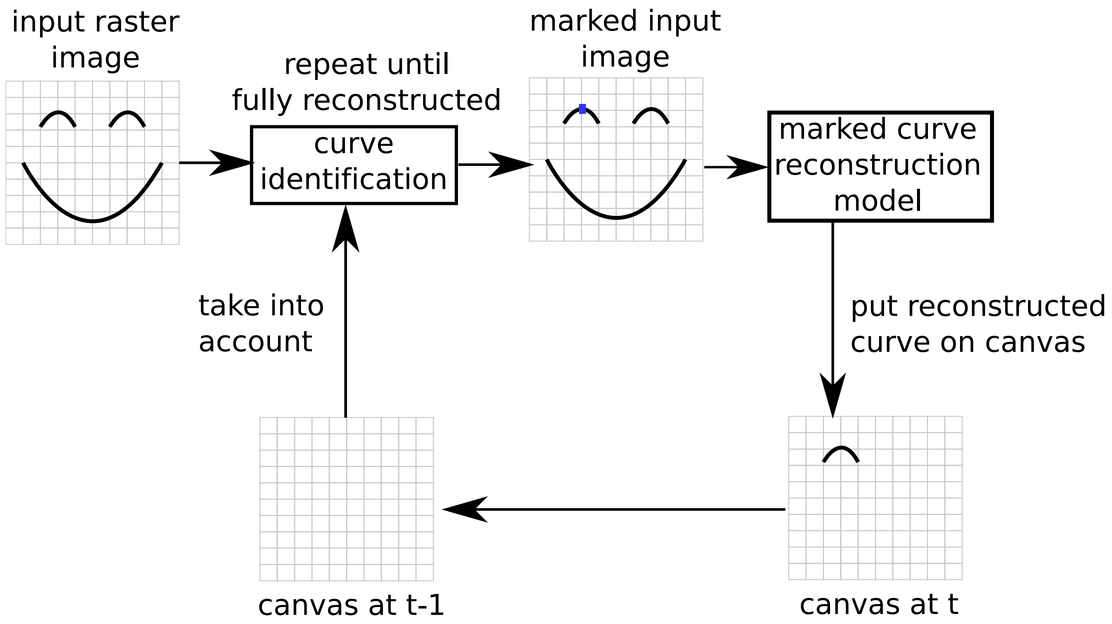
In this section we describe a method to automatically convert line-art raster images into vector images. The method is visualized in Figure 4.2 and consists of two parts: the main part is a learned model that takes as input a raster line-art image and a mark on a curve in this image and outputs a cubic bezier curve which fits the marked curve. The second part is a lightweight algorithm that uses this model iteratively to reconstruct all curves in an image. The marked-curve reconstruction model is described in Section 4.1.1, while the iterative curve reconstruction algorithm is described in Section 4.1.2.

To motivate this architecture, recall that a line-art image consists of a set of bezier curves. The amount of bezier curves is considerably large (see Section 4.2). Following this, the task of line-art image vectorization is decomposed into two non-trivial sub-tasks:

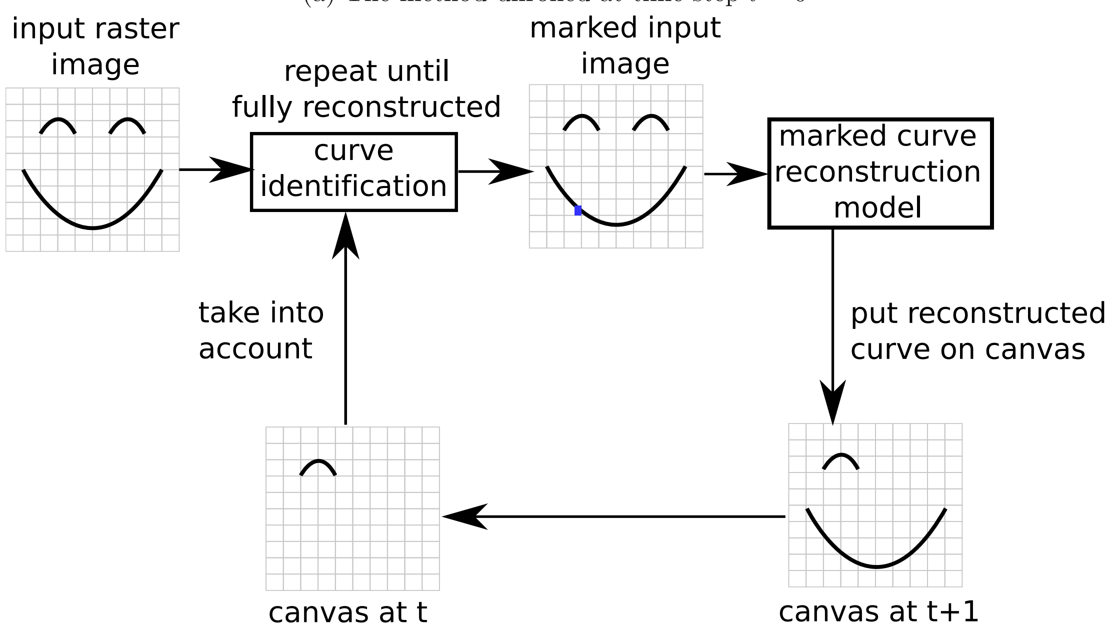
- curve identification: given a line-art raster image and an image of already reconstructed curves (i.e., a *canvas* image), sample a point that lies on a curve (i.e., a *marker*), and
- curve reconstruction: given a line-art raster image and a point lying on a curve (i.e., the marker), reconstruct the marked curve.

Decomposing the task into these two subtasks with a more narrow objective reduces the space of possible solutions of the algorithm, thereby aiding the design of the algorithm. Furthermore, this architecture allows the curve reconstruction and identification to be independent of the number of curves, further decreasing the solution space. Additionally, this structure is more amenable to manual fixing of the output (as described in Sections 1.3 and 2.2.1), since missing curves can easily be reconstructed by invoking the curve reconstruction part with a marker on the curve in question.

Of the two subtasks, curve reconstruction is the more complex part and is handled by the learned marked-curve reconstruction model introduced in Section 4.1.1. On the other hand, curve identification is considerably easier to solve for the data primarily considered in this work (i.e., clean line-art raster images). The curve identification algorithm is described as part of the iterative curve reconstruction algorithm in Section 4.1.2 and simply samples a pixel belonging to a curve of a grayscale line-art raster image. Since the background is white and the curves are colored, this pixel will be black (i.e., closer to 0 than to 1) in a grayscale version of the line-art image. Notice that this curve identification algorithm is both tailored to clean line-art images and not differentiable. Hence, if the input image is in a different domain or a fully differentiable algorithm is needed (such as



(a) The method unrolled at time step $t = 0$



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

Figure 4.2: 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 bezier curve parameters. This output is added to a canvas, which is taken into account when identifying the curve to reconstruct at $t + 1$.

in the cross-domain line-art image vectorization proposed as an extension of this work in Section 1.3), it is necessary to replace the proposed curve identification algorithm with a more suitable alternative. While this is an orthogonal problem, Section 4.1.2 also describes a potential alternative.

4.1.1 Marked-Curve Reconstruction Model

This section details the architecture of the marked-curve reconstruction model, which is depicted in Figure 4.3. The model takes as input a line-art raster image with a mark placed on a curve in it, and outputs the bezier curve parameters fitting the marked curve. This model 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. As an example of a widely used model architecture that follows this principle, diffusion models [Sohl-Dickstein et al., 2015, Rombach et al., 2022] attempt to accomplish image generation by iteratively taking small denoising steps instead of generating the whole image at once.

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. Since the amount of curves in clean frame images is quite high (see Section 4.2), this significantly reduces the space of possible solutions of the model. The other two decisions are based on the input and the output of the model and are explained below.

Input and Output

The input of the model is a line-art raster image. Additionally, this image contains one pixel of a different color from the curves and the background lying on a curve. This pixel is the marker indicating which 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. This way, the model can be trained on the assumption that the center pixel has to lie on the reconstructed curve, avoiding the need for the model to learn to reconstruct the curve at the correct location. Furthermore, note that the centering of the input image on the mark obviates the need to provide the mark location explicitly to the model, since it will be on the center for all input images and is thus implicitly provided. This includes both appending the mark location to the input vector and displaying the mark using a different color on the input image. Hence, the depiction of the mark in the raster image is kept purely for illustrative purposes.

The raster input images are represented using the RGB color model, i.e., each pixel is represented using three numbers in $[0, 255]$. In order to not let multiplications and gradients in the architecture explode, the numbers are divided (i.e., scaled) by the maximum 255 to be in $[0, 1]$. Furthermore, the model is trained and evaluated using clean line-art images only, i.e., images which can be binarized into black and white images, where the curves are black and the background is white. These images could

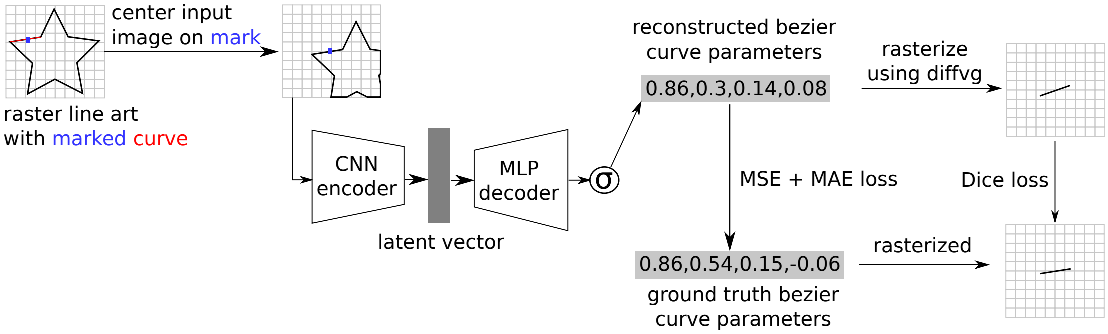


Figure 4.3: Architecture overview of the marked-curve reconstruction model. Note that for brevity, lines with two points are shown instead of cubic bezier curves with four points.

be represented using a single color channel per pixel, which would slightly reduce the model size. However, as there was no difference in model performance between RGB and monochrome input images, the input images are kept in RGB format. This way, no assumption of monochrome input images is baked into the model and it can also handle non-monochrome input images.

The output of the model is defined as the parameters of a cubic bezier 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. Recall that clean frames consist of quadratic and cubic bezier curves with fixed stroke width of predefined colors, as described in Section 2.2.1. Furthermore, the restrictive nature of the output structure reduces the task complexity in three ways. Firstly, the model does not need to learn to use different primitives other than the cubic bezier curve. This can be achieved since clean frames only consist of quadratic and cubic bezier curves, and the possibility of representing quadratic bezier curves as cubic bezier curves. Secondly, the model does not need to learn the correct stroke width of the reconstructed curve, since it is a constant that can be defined for the whole image. Thirdly, the model does not need to learn the correct color of the reconstructed curve, since color follows a predefined schema that can be handled by preprocessing the image.

Model Architecture

The architecture of the marked-curve reconstruction model is depicted in Figure 4.3. Due to the nature of the task requiring the model to generate complex output based on high-dimensional input, it is designed as an encoder-decoder architecture. That is, it consists of an encoder neural network that turns the input image $\mathbf{x}$ into a latent vector $\mathbf{z}$ of predefined length L, and a decoder neural network that turns this latent vector into cubic bezier curve parameters $\mathbf{o}$. In general, the model is designed to be as small and simple as possible and follows standard practices. Note that a small model size has considerable benefits, such as faster computation and less memory requirements, while aiding regularization (see Section 2.3.1).

Since the input is an image, the encoder is a convolutional neural network. As described in Section 2.3.3, convolutional layers have an inductive bias for images. Furthermore, they are locally connected to the input and therefore work for varying input image sizes (i.e., resolutions). The encoder consists of 6 blocks, where each block is formed by a 2-dimensional convolutional layer, followed by 2-dimensional batch normalization and ReLU activation. The architecture is designed following standard practices such that the image size is halved and the channel size is doubled at every convolutional layer.

A disadvantage of using stacked convolutional layers is that the output is a 3-dimensional matrix, which does not satisfy the structure needed as latent vector to be input to the decoder. In order to arrive at a latent vector of predefined length L independent of the input image size, the global pooling technique (see Section 2.3.3 is used). That is, the last convolutional layer has a filter size corresponding to the latent vector length L. Following this, a global average pooling layer is used to reduce the space dimensions of the resulting output, leading to a 1-dimensional latent vector of length L.

The hyperparameters of the encoder layers are displayed in Table 4.1. Each convolutional layer doubles the filter size of the previous layer and has a stride of 2 and a padding of 1. The last convolutional layer has a stride of 1. Note that the batch normalization following each convolutional layer is not displayed. They follow the same size as their preceding convolutional layer output and are parameterized with $\epsilon = 10^{-5}$ and a momentum of $\mu = 0.1$.

Note that, as described above, the encoder architecture is designed to handle variably sized input, with these variables being denoted in Table 4.1. The batch size B is used to process multiple observations in parallel and increase the effectiveness of batch normalization by decreasing the variance (see Section 2.3.1). 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 $L = 128$. Note that the width and height do deliberately not correspond to the exact resolution of clean animation frames in the dataset (see Section 4.2). This is done to show that the model does not overfit to a specific resolution. As an aside, CNNs are typically trained on significantly smaller W and H [He et al., 2016a], especially when trained for image classification. However, these small image resolutions would compress the clean animation frame beyond a reasonable possibility of detecting individual curves.

The decoder is a 2-layer MLP, which turns the latent vector of length L into a vector of length $P * 2$, where P is the number of cubic bezier curve parameters. Since cubic bezier curves are parameterized by a start point, an end point and two control points, $P = 4$. With only one hidden layer, the decoder is as shallow as possible. The hidden layer is introduced to enable the decoder to learn non-linear transformations of the latent vector. It is followed by batch normalization, which is parameterized with $\epsilon = 10^{-5}$ and

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

a momentum of $\mu = 0.1$ as in the encoder and a ReLU activation. The output layer outputs a vector of length $2P$. Intuitively, this could directly be used as output. However, these numbers are unbounded and could theoretically go towards ∞. Hence, the output is restricted to $[0, 1]$ using the sigmoid activation function (see Equation (2.1)). The x coordinates of the cubic bezier curve points are then scaled with the image width, while the y coordinates are scaled with the image height.

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

Table 4.2: Summary of the layers of the encoder neural network of the marked-curve reconstruction model.

Tables 4.1 and 4.2 show the numbers of learnable parameters of the model. In total, the model has 2,169,672 learnable parameters. The distribution of these learnable parameters indicates that the model is encoder-heavy, with a large portion of them assigned to the convolutional layers. Since the encoder does the heavy lifting, more complex encoder architectures such as the ResNet [He et al., 2016a] and ConNeXt [Liu et al., 2022] were tried as alternatives. For these architectures, encoder weights which were pretrained on larger image datasets such as ImageNet [Russakovsky et al., 2015] are available, obviating the need to train an encoder from scratch. However, they did not perform significantly better than the simple encoder architecture introduced here. Hence, they are not used in order to keep the model as small and simple as possible.

Training

The model is trained to generate a curve that resembles the ground truth (i.e., the gold standard) curve both visually and in its semantic topology. For this reason, a combination of a raster-based loss for visual similarity and a vector-based loss for semantic correctness is used to train the model. Intuitively, the raster-based loss is used to optimize the model to output a curve that covers the pixels of the original line as closely as possible. The vector-based loss is then used to optimize the model to output semantically correct curve parameters. Furthermore, the vector loss is multiplied with a weight of 100 to not let the raster-based loss dominate the combined loss. This task-derived loss combination is an important distinction from related work [Reddy, 2021, Mo et al., 2021, Egiazarian et al., 2020].

The vector loss follows [Egiazarian et al., 2020] and is an even combination of MAE and MSE. It is defined in Equation (4.1), where $\mathbf{o}$ is the model output, $\mathbf{y}$ are the ground truth cubic bezier curve parameters and L_1 and L_2 are defined in Equations (2.5) and (2.8), respectively. The loss is closely related to the Huber loss defined in Equation (2.12). Intuitively, since the MAE and the MSE have their advantages and disadvantages, the loss simply uses both using the same weight (see Section 2.3.2).

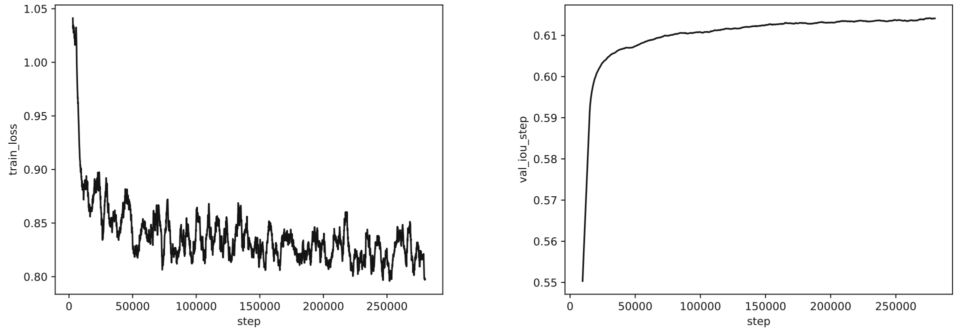
$$L(\mathbf{o}, \mathbf{y}) = 0.5 * L_1(\mathbf{o}, \mathbf{y}) + 0.5 * L_2(\mathbf{o}, \mathbf{y}) \tag{4.1}$$

Defining a raster-based loss is more difficult, since the model outputs the cubic bezier curve in vector format, which needs to be rasterized. As described in Section 2.1.2, rasterization is trivial and can be done deterministically. However, for the loss it is crucial that the rasterization is differentiable (see Section 2.3.1). Hence, the differentiable rasterizer introduced by Li et al. [2020] is used to rasterize the cubic bezier curve parameters. This raster output image is then compared to the raster input image, with all curves aside from the marked curve removed.

A typical choice for comparing two raster images is to simply use the MSE (or, less commonly the MAE). However, the rasterized curve image exhibits a significant class imbalance, since most of the image is white with only the curve being black. Hence, as described in Section 2.3.2, the dice loss defined in Equation (2.16) is a better choice than MSE. Accordingly, the performance of the model is evaluated using the IoU defined in Equation (2.10) of the output and the ground truth curve raster image.

The model is trained using the widely used Adam [Kingma and Ba, 2015] optimizer with a learning rate of $\eta = 5 * 10^{-4}$. The other hyperparameters are set to the default PyTorch [Paszke et al., 2019] values, i.e., no weight decay, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 * 10^{-8}$.

To visualize the training process, figure Figure 4.4a shows the loss per iteration, where one iteration indicates one batch of the data being processed. After all batches are processed, the training continues again starting from the first batch. The batch size is set to $B = 32$, which is the largest size possible given available dedicated GPU memory. Figure 4.4b shows the performance of the model on the validation dataset provided by Tonari Animation (described in Section 4.3.1). The performance is measured using the IoU of one reconstructed curve and the ground-truth curve (see Section 4.4.1). It converges to a single curve IoU of 0.62.



(a) The training loss. (b) The validation single curve IoU.

Figure 4.4: Training progress of a model measured using the loss on the training dataset and the single curve IoU on the validation dataset per iteration.

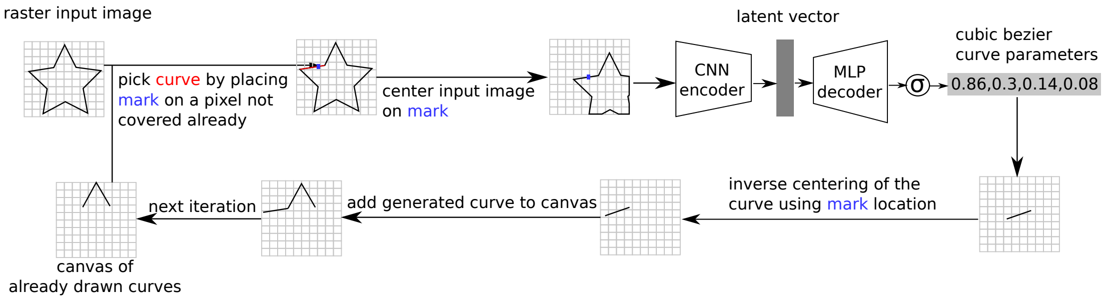


Figure 4.5: Overview of the iterative curve reconstruction algorithm. This overview goes into more detail regarding the curve identification and reconstruction than Figure 4.2.

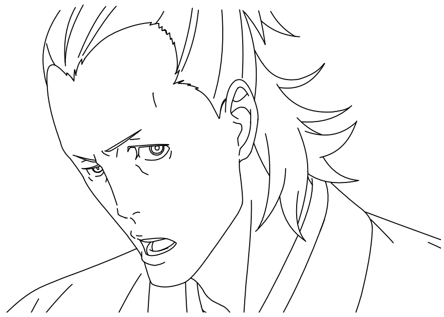
4.1.2 Iterative Curve Reconstruction Algorithm

The marked-curve reconstruction model introduced in Section 4.1.1 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:

1. handles color (and stroke width) information,
2. given the input image and a canvas of already reconstructed curves, computes markers identifying curves to reconstruct, and
3. invokes the marked-curve reconstruction model using the input image and a marker of an identified curve and places the output on the canvas.

The algorithm is depicted in Figure 4.5 and explained in the following.

(a) Full clean animation frame.



(b) Black segment of the clean animation frame.

(c) Blue segment of the clean animation frame.

(d) Red segment of the clean animation frame.

Figure 4.6: Example of a clean animation frame provided by Tonari Animation segmented by color.

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 (see Section 2.2.1), it is necessary for the algorithm to produce the correct color information for all predicted curves. This can be done using simple pre-processing and post-processing steps, which both reduces the task complexity of the model and ensures that the output is correct.

In detail, note that the color schema of clean animation frames is known a priori, as defined in Figure 2.10. Hence, it is possible to segment the input image according to these colors (see Section 2.3.2). Figure 4.6 exemplifies this. Then, the curve colors of each segment are set to black and each segment is individually input into the marked-curve reconstruction model. The output of the marked-curve reconstruction model contains no color information, but since the true color of the segment is known, the color of the output can be set to the segment color.

In the same vein, the marked reconstruction model does not output stroke width information. However, since all curves in a clean animation frame share the same stroke width, it suffices to define one constant stroke width for the input image and to apply this to all reconstructed curves. The stroke width can either be defined by the user or inferred from the input image.

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 4.1.2) 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, where a pixel is considered black if it is closer to 0 than to 1. This pixel is guaranteed to lie on a curve. As an aside, a potential improvement to this algorithm is to always sample the pixel from the largest contiguous area of black pixels, in order to reconstruct the longest curves first. However, this has not been implemented in order to keep the algorithm as simple as possible.

In order to ensure that only curves that have not yet been reconstructed are identified, the canvas image is subtracted from the input image whenever a new curve is added to the canvas and before the marker pixel is sampled. This also helps the marked-curve reconstruction model to not output duplicate curves.

Both the curve identification by sampling black pixels and the subtraction of the canvas image from the input image have the assumption of clean line-art images. Thus, when the line-art vectorization is applied to input images of different domains, these two parts have to be adapted. If there exists a binarization algorithm of considerable quality for this domain (e.g. Su et al. [2010]), then simply binarizing images with this algorithm will suffice. Otherwise, one alternative is to train a curve identification model. This alternative would render the entire line-art image vectorization method end-to-end differentiable. The curve identification model would take as input both the input line-art image and the canvas image, where the latter could be appended as a fourth color channel to the former. The output would a vector containing two elements, constrained by the sigmoid function to lie in $[0, 1]$, indicating the coordinates of the marker. Then, the architecture would be a convolutional neural network with global average pooling on a filter size of 2 at the end. There are multiple candidates for loss functions. One possibility is to calculate the distance of the marker to all ground truth curves that are not present in the canvas image and taking the minimum. This would require the training images to also be available in vector format.

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 4.1. In this algorithm, the remaining image is the raster image containing curves that are not yet reconstructed. The canvas image is stored as vector image using the SVG format. It is rasterized when being used to update the remaining curves image.

Note, that Line 5 in Algorithm 4.1 constitutes an intuitive stopping criterion enabled by the progressive canvas image subtraction from the remaining image. The curve

Algorithm 4.1: Iterative Curve Reconstruction.

```
Input: A raster line-art image.
Output: A vector line-art image.
Segment input image by color;
foreach image segment do
    canvas = an empty vector image of the same size as the input image;
    remaining = image segment;
    while number of black pixels in remaining > T; do
        Compute marker by applying curve identification on the remaining image;
        Centered image = center the remaining image on the marker;
        reconstructed curve = invoke the marked-curve reconstruction model
          using the centered image;
        Inverse the center location of the curve by using the mark location;
        Add the reconstructed curve to the canvas image;
        remaining = remaining - rasterized canvas image;
    end
    Set color of all curves in the canvas image to the segment color;
end
Merge the canvas images;
return Merged canvas images
```

reconstruction is iteratively applied until the number of black pixels (i.e., the number of possible markers) in the remaining image is greater than some threshold T. If the curve identification and reconstruction worked perfectly, the difference between the rasterized canvas image and the remaining image would reach 0 at some point. In this case, the threshold should be set to $T = 1$. However, since errors in the reconstructed curves are to be expected, there will remain a number of black pixels that are part of an already reconstructed curve that does not fully cover the curve in the input image. Repeatedly invoking the marked-curve reconstruction model on such pixel artifacts will lead to worse results. Since missing a few curves is not a significant issue, it is tolerable to set the threshold to a low number of black pixels greater than 1. For this algorithm, the threshold is set to $T = \lfloor B * 0.1 \rceil$, where B is the number of black pixels in the original image.

Advantages and Limitations

The theoretical runtime complexity of the algorithm as defined in Algorithm 4.1 is in $\mathcal{O}(p)$, where p is the number of black pixels in the input image, which is connected to the number of curves n. As the performance of the marked-curve reconstruction model increases, the runtime will be approximately linear in n. However, it is possible to significantly reduce the runtime by processing a batch of curves $b < n$ concurrently, instead of one curve at a time. For this, the curve identification algorithm described in Section 4.1.2 can be adapted to sample b black pixels instead of a single black pixel. To decrease the probability of multiple marks belonging to the same curve being sampled, the image should be divided in b patches, with a single mark being sampled in each patch. The marked-curve reconstruction model does not need to be adapted, since it possesses the ability to processes a batch of inputs, which is already done at training time. Note that this adaptation increases the memory requirement by roughly b times. However, since the test dataset is rather small (see Section 4.2), this parallel version of the algorithm has not been implemented.

Furthermore, an advantage of this algorithmic structure is that it significantly eases the effort required to manually fix an output image. While identification of missing curves is a trivial task for humans, fixing wrongly reconstructed curves is tedious. Following this, the algorithm is designed to maximize the former in lieu of the latter. On the one hand, it allows to post-hoc reconstruct curves that were missed by the first run of the algorithm without affecting the rest of the output by simply placing a mark on a random point of the curve in question and invoking the marked-curve reconstruction model. On the other hand, by letting the learned model focus on reconstruction of a single curve, its reconstruction results will be at least better than a model that needs to perform reconstruction and identification of all curves, thereby reducing the need to fix reconstructed curves.

Lastly, there are two issues associated with overlapping curves, i.e., pixels which are part of multiple curves. Firstly, this directly affects the iterative curve reconstruction algorithm. Suppose the marked-curve reconstruction model outputs one of the multiple overlapping curves, then this curve will be placed on the canvas image and removed from the input image. In this process, the overlapping pixels will also be removed, leaving behind a hole in the remaining curves, effectively splitting them. Hence, if the hole is large enough, the curve reconstruction model will reconstruct only one part of the split curve instead of the whole curve. Secondly, the curve reconstruction model will receive ambiguous information on which curve to reconstruct during training in these cases, hampering its ability to derive meaningful gradients at all in the worst case. However, both of these issues are negligible since the amount of overlapping curves is small, as the calculation in Section 4.2 shows.

4.2 Dataset

The dataset used to train and evaluate the line-art vectorization method consists of two parts: a human-generated and a synthetic dataset. Both datasets contain clean line-art images as vector images using the SVG file format, with a uniform color for the background (white) and the curves (black). An example indicating the vector primitives used can be seen in Figure 4.7, with each primitive represented using a mutually exclusive color. The first part is described in Section 4.2.1 and consists of line-art images drawn by professional and amateur artists. The synthetic part is described in Section 4.2.2. Section 4.2.3 notes irregularities and limitations of the assembled dataset. Steps taken to deal with these irregularities and further processing steps are described in Section 4.2.4.

	images	images with overlapping curves
tonari	139	138
sketchbench	425	-
tuberlin	20000	-

Table 4.3: Summary of the subsets of dataset.

4.2.1 Human-generated Dataset

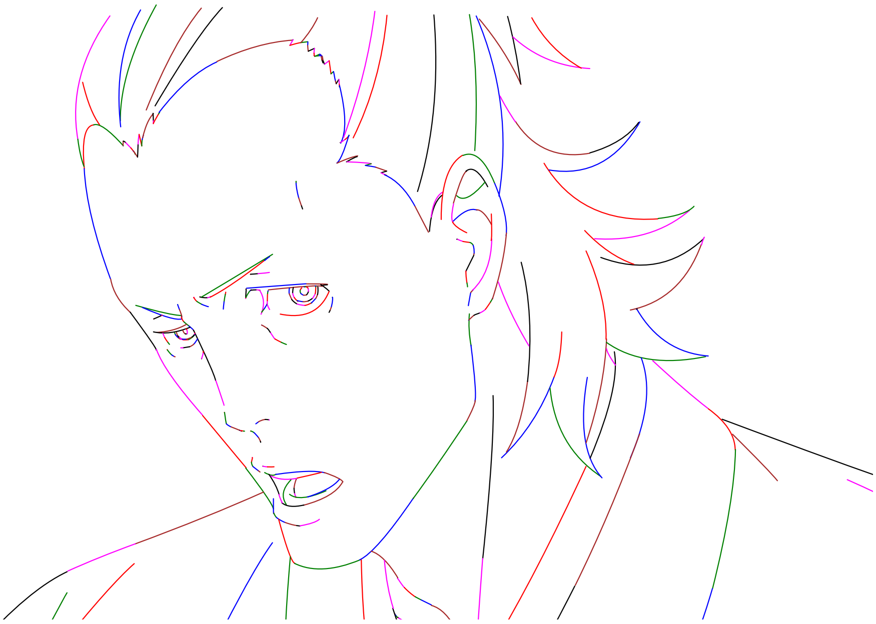


Figure 4.7: A clean animation keyframe in vector format with all outline (i.e., black) curves indicated by alternating colors. Original keyframe provided by Tonari Animation.

The human-generated data is the most important part of the dataset, since it enables the marked-curve reconstruction model to reconstruct curves in a semantically meaningful way. It consists 20564 vector images from three sources displayed in Table 4.3. As noted in Section 1.3, a small dataset of 139 real-world clean animation frames is provided by Tonari Animation, which is detailed in Section 4.2.1. While it forms the heart of this dataset, the amount of data is rather limited, necessitating additional sources of data. Two additional sources are used: One of these comes from a sketch cleanup benchmark by Yan et al. [2020] and is explained in Section 4.2.1. Since this is also rather limited in size, a collection of amateur sketches by Eitz et al. [2012] is used as additional dataset of medium-to-low quality but large quantity.

Figure 4.8: An example of a clean animation frame from a sample animation sequence depicted in Figure 2.8, provided by Tonari Animation.

Tonari Clean Animation Frames

Figure 2.7b shows one of 194 clean animation frames in raster format provided by Tonari Animation for an earlier work by Kugler [2023]. Since they are not vector images, they cannot be used for this work. However, for 50 of these clean animation frames, the original Clip Studio [CELSYS, Inc., 2021] files used to draw the images were still available, enabling access to the underlying vector primitives. Additionally, Clip Studio files of 4 sample animation sequences were provided, from which 89 clean animation keyframes and inbetweens could be extracted. An example of this additional data can be seen in Figure 4.8. In total, this leads to a dataset of 139 clean animation frames in vector format provided by Tonari Animation. Note that inbetweens are very similar to each other (see Section 2.2), thus containing a high amount of redundant information, limiting their usefulness to train the model. Still, due to the low amount of available data, all inbetweens are used.

Since the Clip Studio files provided by Tonari Animation are in a format which cannot be used to train or evaluate the marked-curve reconstruction model, the vector primitives have to be manually extracted from these files. For this, the trial version of Clip Studio version 1.10.5 [CELSYS, Inc., 2021] was used on a Windows 10 machine. Using the Export Vectors function, each of the 139 clean animation frames was exported as SVG file. Figure 2.11 shows the result of this extraction on the example in Figure 2.7b. The color of the curves is preserved according to the schema defined in Figure 2.10. This information is used by the iterative curve reconstruction algorithm as noted in Section 4.1.2. Note that only curves could be exported, with filled color regions missing from the output. However, as mentioned in Section 2.2.1, they can trivially be added by bucket filling and are thus not an integral part of the clean animation frame. Hence, only the curves are considered for this work.

It is important to note that while images are available in vector format and can be theoretically displayed at all resolutions, they are generally drawn with a target resolution of 720x405px by Tonari Animation.

SketchBench

The rough sketch cleanup benchmark by [Yan et al., 2020] provides public high-quality line-art vector images. These are sourced from a varied and balanced amount of domains and artists and processed according to a workflow depicted in Figure 4.9. First, 151 sketches of the genres freeform, fashion, products, logos and architecture drawn by various artists are retrieved online. These sketches were drawn on paper or digitally and are only available in raster format. Then, up to 3 out of 7 contracted artists per sketch trace clean vector versions of the rough sketch raster image, yielding 425 SVG images. These traced vector images contain curves indicating shadows, textures and scaffolds, which are removed to produce clean vector versions of the original rough sketch raster image. Furthermore, the curves are normalized to a constant stroke width.

Only clean line-art images are used to train the marked-curve reconstruction model. Hence, the rough raster images and the traces before cleanup are removed, with only the normalized and cleaned versions of the traced sketches remaining in dataset. These images are indicated by the `norm_cleaned.svg` suffix of their filename. In total, this leads to a dataset of 425 vector images.

The dataset resembles the Tonari clean animation frames closely, containing only quadratic and cubic bezier curves with a constant stroke width. However, it differs in some respects. Out of the images, sketches of the freeform genre are overrepresented and most closely resemble clean animation frames. However, due to a general lack of data, sketches of all genres are used for this work. Furthermore, there are only black curves in this dataset. Hence, the segmentation noted in Section 4.1.2 does not need to be performed on this data.

TU Berlin amateur sketch collection

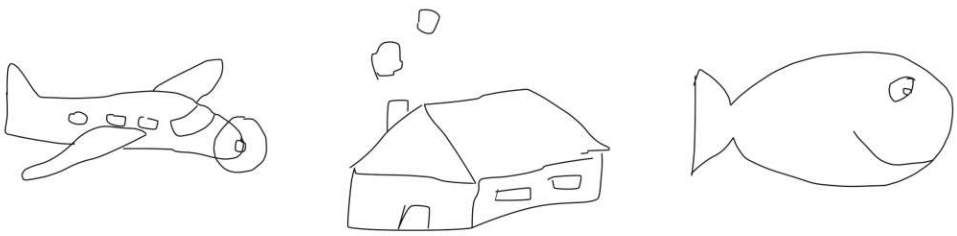
In order to increase the size of the human-generated dataset, the amateur sketch collection by Eitz et al. [2012] is also used. The sketches were collected by TU Berlin for the purpose

(a) A rough sketch drawn on paper as raster image.

(b) Full vector tracing of the original rough sketch.

(c) Normalized and cleaned version of the vector trace.

Figure 4.9: Available versions of an example freeform sketch in the SketchBench dataset [Yan et al., 2020].



(a) An amateur sketch of the airplane category.

(b) An amateur sketch of the house category.

(c) An amateur sketch of the fish category.

Figure 4.10: Example of the TU Berlin amateur sketch collection by Eitz et al. [2012].

of creating a classification model. Hence, the dataset contains in total 20,000 amateur sketches of 250 pre-defined categories. The categories are derived from the LabelMe dataset [Russell et al., 2008] and contain objects such as airplanes, houses, and fishes. Examples are displayed in Figure 4.10. The sketches are drawn by 1,350 amateur artists contracted using Amazon Mechanical Turk. They contain only outlines and are drawn without a reference image, with a median drawing time of 86 seconds. The images are also available in the SVG format, which can be readily used for this method.

There exist other collections of amateur sketch vector images such as Quick, Draw! [Ha and Eck, 2018]). These could also be added in order to increase the overall dataset

		tonari	sketchbench	tuberlin
paths	median	495.00	44.00	13.00
	IQR	210.00	96.00	15.00
curves	median	1476.00	156.00	78.00
	IQR	1041.00	381.00	72.00
curves / path	median	1.00	2.00	4.00
	IQR	2.00	1.00	4.00
path length	median	65.61	219.82	176.01
	IQR	29.90	327.81	223.20
curve length	median	9.60	67.82	32.17
	IQR	6.83	106.37	19.42
overlap curves	median	10.00	-	-
	IQR	10.00	-	-

Table 4.4: Summary statistics for the human-generated subsets of the dataset

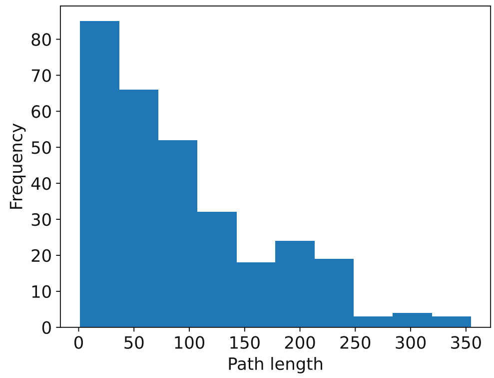
size. However, the TU Berlin dataset is already dominating the other, high-quality sources by number of images. If the Quick, Draw! dataset was added, the marked-curve reconstruction model would be even more biased to amateur sketches, to the detriment of performance on professional line art. Hence, other amateur sketch collections are not considered in this work.

Statistics

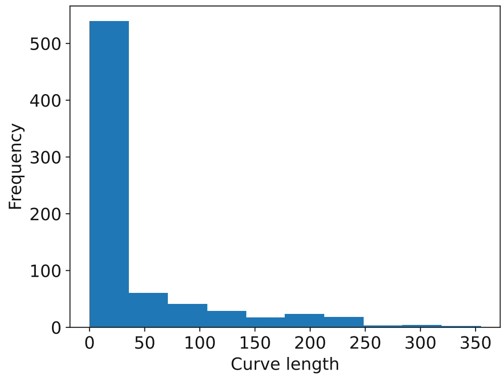
All images in the human-generated dataset are clean line-art images in vector format. The images contain a uniform background and cubic bezier curves, which are grouped together to form *paths*. Table 4.4 shows statistics of the dataset and its subsets. Note that the median is used as robust location estimate and inter-quartile range (IQR) as robust scale estimate. This is due to the assumption that there are outliers in the data, which can have an outsized influence due to the limited size (excluding the TU Berlin subset). Figure 4.11 shows the left skewed nature of distributions of the example clean animation frame depicted in Figure 4.6a and corroborates this assumption.

The statistics displayed in Table 4.4 help with deciding on which level of graphic primitives the method should be based. There are two options: basing the method on *curves*, which gives more flexible and fine control over the vector output and reduces the complexity for the model by only requiring to reconstruct a single primitive, or *paths*, which can be considered more semantically meaningful but would require the model to reconstruct a group of curves at a time.

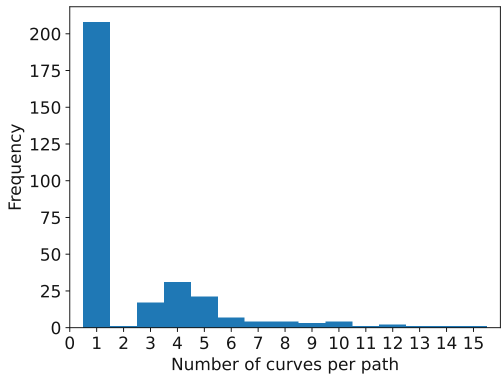
Table 4.4 shows that the average number of curves per path for the high-quality subsets is very small. To exemplify this, Figure 4.11c shows the distribution of the number of curves per path in the image depicted in Figure 4.6a. There seems to be a bimodal distribution,



(a) Histogram of the length of each path.



(b) Histogram of the length of each curve.



(c) Histogram of the number of curves per path.

Figure 4.11: Statistics of the clean animation frame depicted in Figure 4.6a.

with most paths either consisting of a single curve or more than 2 curves. It is clear that the overwhelming majority of paths belongs to the former, making this higher-level primitive somewhat redundant. Interestingly, both the average and the variance of the number of curves per path in the TU Berlin subset is significantly higher than in the high-quality subsets, indicating that a significant amount of amateur artists made more use of this primitive than professional artists. However, since the focus of the method should lie on the high-quality subsets, it is decided to base the method on curves and to discard path information.

Another important information for the decision between curves and paths is the average length of each, which is measured in pixels. Interestingly, Table 4.4 shows that the average curve length of Tonari clean animation frames is significantly lower than the other two subsets, with SketchBench images having the longest curves on average. The same finding is maintained for the average path length. Furthermore, the ratio of curve length by path length is similar for the Tonari and TU Berlin subsets, with the SketchBench subset having roughly double the ratio. In general, and even for the Tonari clean animation frames, curves seem on average to be long enough to be independently recognized and restored, further corroborating the use of curves as graphical primitive. Still, due to the high variance of curve and path lengths and the seemingly bimodal distribution of the amount of curves per path, using paths as base graphical primitive is also a legitimate avenue.

Furthermore, when looking at the statistics, two peculiarities in the data become apparent. Firstly, Tables 4.3 and 4.4 show that there exists a significant amount of overlapping curves, where two curves are considered overlapping if they have more than one intersection and there is at least one intersection that is not at the start or the end point, with a tolerance of 0.1. This calculation is computationally expensive, as this requires comparing every curve with every other curve. Hence, this information is only provided for the important Tonari subset. Interestingly, the average number of overlapping curves per image is 36 ± 36 with the average number of overall curves per image being 1476 ± 1041. This suggests that most images contain only a little amount of overlapping curves. However, since nearly all images have at least one curve overlapping another, this needs to be accounted for when dealing with these images.

4.2.2 Synthetic Dataset

Since the human-generated part of the dataset is limited in size, the dataset also includes synthetic data. While automatic vectorization of existing clean line-art raster images is not yet possible (as this is the whole purpose of this work), the reverse, i.e., the rasterization of vector line-art images is trivial, as described in Section 2.1.3. Hence, for the synthetic data, line-art vector images are automatically generated and rasterized. However, automatic generation of line-art vector images is challenging. Since automatically generating high-quality line-art vector images such as the Tonari clean animation frames is still an open research question, one has to resign oneself to a lower quality of the generated images. Due to this low quality, the synthetic dataset is combined with the human-generated dataset at a 1 to 5 ratio at train time, i.e., for every training epoch, the human-generated vector images are topped up with synthetic vector images matching a fifth of their amount.

The generation algorithm defined for this work simply produces a low number of random cubic bezier curves for a vector image. The main hyperparameter of the algorithm is the curve amount range, which is set to $[3, 8]$ unless otherwise stated. This means that the algorithm produces at least 3 and at most 8 curves per image, with the actual number being chosen randomly. The generation is restricted to this low amount in order to minimize the probability of overlapping curves, which otherwise would need to be detected and removed in a computationally costly manner. The curves themselves are generated by sampling 8 random numbers independently from the uniform distribution with interval $[0, 1)$. These numbers constitute the coordinates of the 4 points needed to parameterize a cubic bezier curve. Since the synthetic image needs to have the same size as the human-generated images, the x-coordinates and y-coordinates of the random

(a) Image with 3 random cubic bezier curves. (b) Image with 5 random cubic bezier curves.

Figure 4.12: Example images of the synthetic dataset. Note the apparent low quality in comparison with other example images in Figures 4.8 to 4.10.

points are scaled by the image width and image height chosen by the human-generated dataset, respectively.

Figure 4.12 shows examples of synthetic line-art images. It can be seen that they differ in quality from the human-generated dataset. Furthermore, they contain significantly fewer curves. In combination, the curves in the human-generated dataset form recognizable objects, while the random synthetic curves do not represent shapes that can be found in clean animation frames. Hence, it is clear that the synthetic dataset is not similar to the clean animation frames (on which the method is intended to be used on) and is wholly insufficient to train the marked-curve reconstruction model. However, by being used in combination with the human-generated dataset, it can serve another purpose: it provides new learning signals to the model. Importantly, the human-generated dataset contains a limited amount of semantic objects to the visual presence of which the model could theoretically overfit. On the other hand, the synthetic images contain only curves that are *guaranteed* to not be in a semantic relationship with each other. Therefore, the synthetic dataset forces the model to focus on reconstructing a single curve correctly when no semantic visual indications are present. Furthermore, the synthetic data likely contains curve shapes not present in the human-generated data. Fitting the model to this data can be useful when the model is applied to images containing objects and curves that are visually very different from the ones in the human-generated dataset. Keep in mind, however, that this is only beneficial to the model if the semantics-free data forms but a small part of the entire dataset.

To optimize the runtime of the algorithm, a batch of points is directly created on the GPU per invocation. The images are synthesized on-line instead of precomputed, i.e., the generation algorithm is run concurrently to the training process, generating new data for each training iteration. Another possibility is to run the generation algorithm beforehand and to save a large amount of precomputed random images on disk. These could then be loaded at training time. The on-line variant is chosen because it has three advantages: firstly, it ensures that every batch generated consists of new data, preventing overfitting if the model is run for a long time. Secondly, it requires no disk space, while the disk space required for the precomputed images can be considerably large. Thirdly, it makes changes to the generation algorithm easier, since the images do not have to be precomputed after every change. Furthermore, the runtime difference between generating a batch of 8 random numbers directly on the GPU versus loading the images from disk to the GPU is negligible.

4.2.3 Limitations

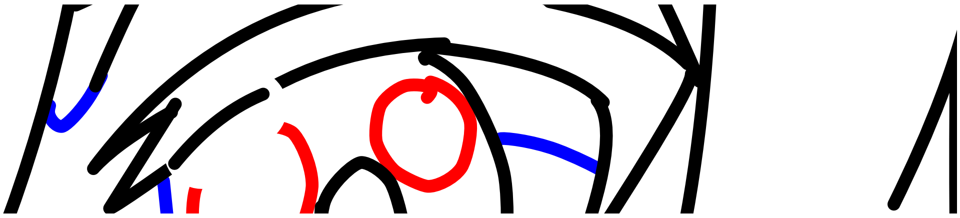
The dataset assembled for this work is of considerable quality, but still comes with a range of limitations. It is important to note that these significantly affect the model training and the evaluation.

Since publicly available clean animation frames in vector format are rather limited, the amount of high-quality data in the dataset is considerably small. This is a major limitation of the dataset, as training a deep learning model usually requires a large amount of data. While the synthetic data introduced in Section 4.2.2 increases the dataset size significantly, it is not of sufficient quality to effectively alleviate this limitation. There exist some other publicly available high-quality data, such as sample data of open animation tools like OpenToonz [DWANGO Co., Ltd., 2023]. However, the amount of vector images in these samples is very limited. Additional sample data exists for proprietary animation tools like CACANi [CACANi Pte Ltd., 2022], which can not be used without the tool and are similarly few in number. Another possibility is to extend the dataset with sources from different domains such as product design sketches [Gryaditskaya et al., 2019]. This was not considered in this work beyond the SketchBench subset due to the focus on clean animation frames.

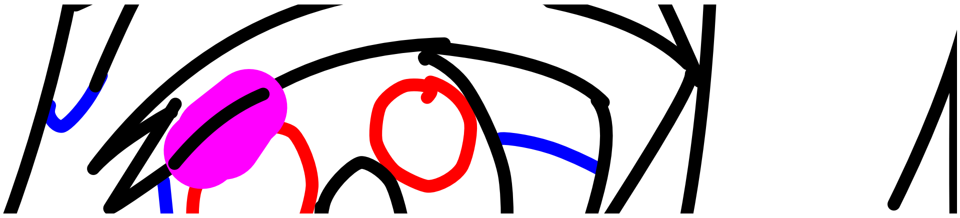
Another limitation stems from the assumption that different artists draw vector images using slightly different graphical primitives and topologies. Hence, the distribution of artists in this dataset affects the performance of the line-art vectorization method. This distribution is rather skewed in the Tonari subset. Out of this subset, the initial 50 images were drawn by a single artist, while the artist information about the 89 keyframes and inbetweens is unknown. On the other hand, the distribution of artists in the SketchBench and TU Berlin subsets are rather even, which renders them helpful in preventing possible overfit to the artist distribution in the Tonari subset.

Irregularities

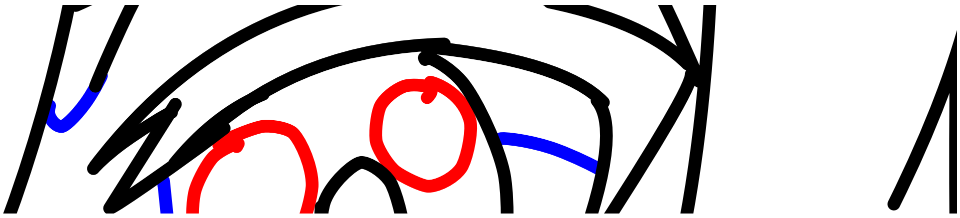
Furthermore, there are various irregularities in the Tonari clean animation frames that need to be accounted for when processing them. The biggest is that there are a number



(a) The vector image rasterized normally, zoomed into the right upper eye region.



(b) The vector image rasterized normally with the color of the white patch set to magenta, thus making it visible.



(c) The vector image rasterized with the white patch removed, rendering the underlying incorrect curve visible.

Figure 4.13: An example of a white patch being used to correct incorrect curves. While the patch and the incorrect curve is not visible in the rasterized image, it is still present in the vector structure of the image. The full clean animation frame can be seen in Figures 2.7b and 2.11.

of apparently incorrectly drawn curves in the images. These are ostensibly erased by drawing a white curve with high stroke width over them. Figure 4.13 shows an example of such a patch, which is not visible when rasterized, but is still present in the underlying vector topology. Another irregularity is that curves often extend outside the viewbox, as depicted in Figure 4.14.

Additionally, some images contain a significant amount of overlapping curves, as noted in Table 4.4. Tables 4.3 and 4.4 show that Tonari clean animation frames contain 39 ± 39 overlapping curves on average, with almost all images containing overlapping curves. As an example, the image shown in Figure 4.8 contains 428 overlapping curves. This includes both curves with a long overlap, as displayed in Figure 4.15a and curves with a minor overlap, as displayed in Figure 4.15b. As noted in Line 16, overlapping curves can lead to incorrect outputs of the line-art vectorization method.

Altogether, these irregularities are not apparent once the vector images are rasterized,

(a) The vector image rasterized using the specified view box.

(b) The vector image rasterized using a view box of 720x465px.

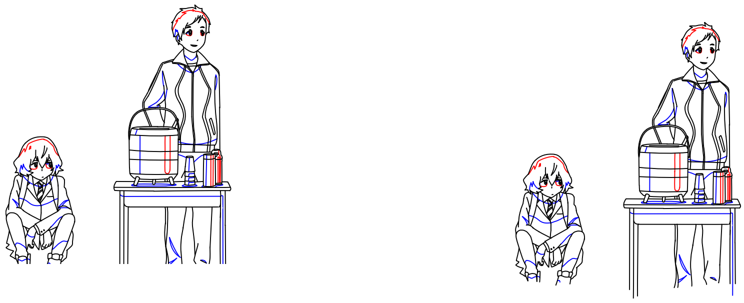
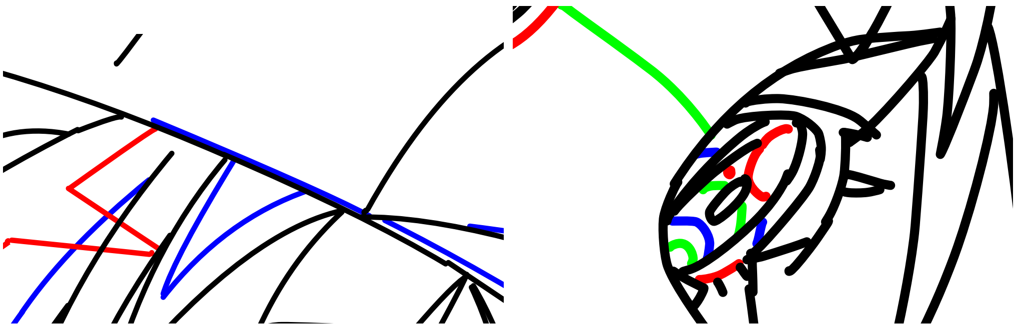


Figure 4.14: An example of a clean animation frame with curves extending outside of the specified view box of 720x405px. Notice the extended curves at the bottom. Original clean animation frame provided by Tonari Animation.



(a) The frame depicted in Figure 4.8 zoomed in to the cap region showing a black and a blue curve in the middle with a long overlap.

(b) The frame depicted in Figure 4.8 zoomed in to the right eye, showing two black curves on the left of the eye with a minor overlap.

Figure 4.15: Examples of overlapping curves in the clean animation frame displayed in Figure 4.8.

but are still problematic when the vector structure of the images is used for training and evaluating the method.

There are also two irregularities that are visible both in vector and in raster format. One of them can be seen when considering the color distribution of paths shown in Table 4.5. As only the Tonari clean animation frames contain colors other than black, Table 4.5 only shows information about this subset. It can be seen that there is a considerable amount of paths of a different color than the ones allowed according the the schema defined in Figure 2.10. White is the most present color that is not in the schema, which can be explained with the white patches displayed in Figure 4.13. The other colors not allowed by the schema are only used by less than 200 paths. It can safely be assumed that these are artifacts of artists choosing a slightly wrong color while drawing.

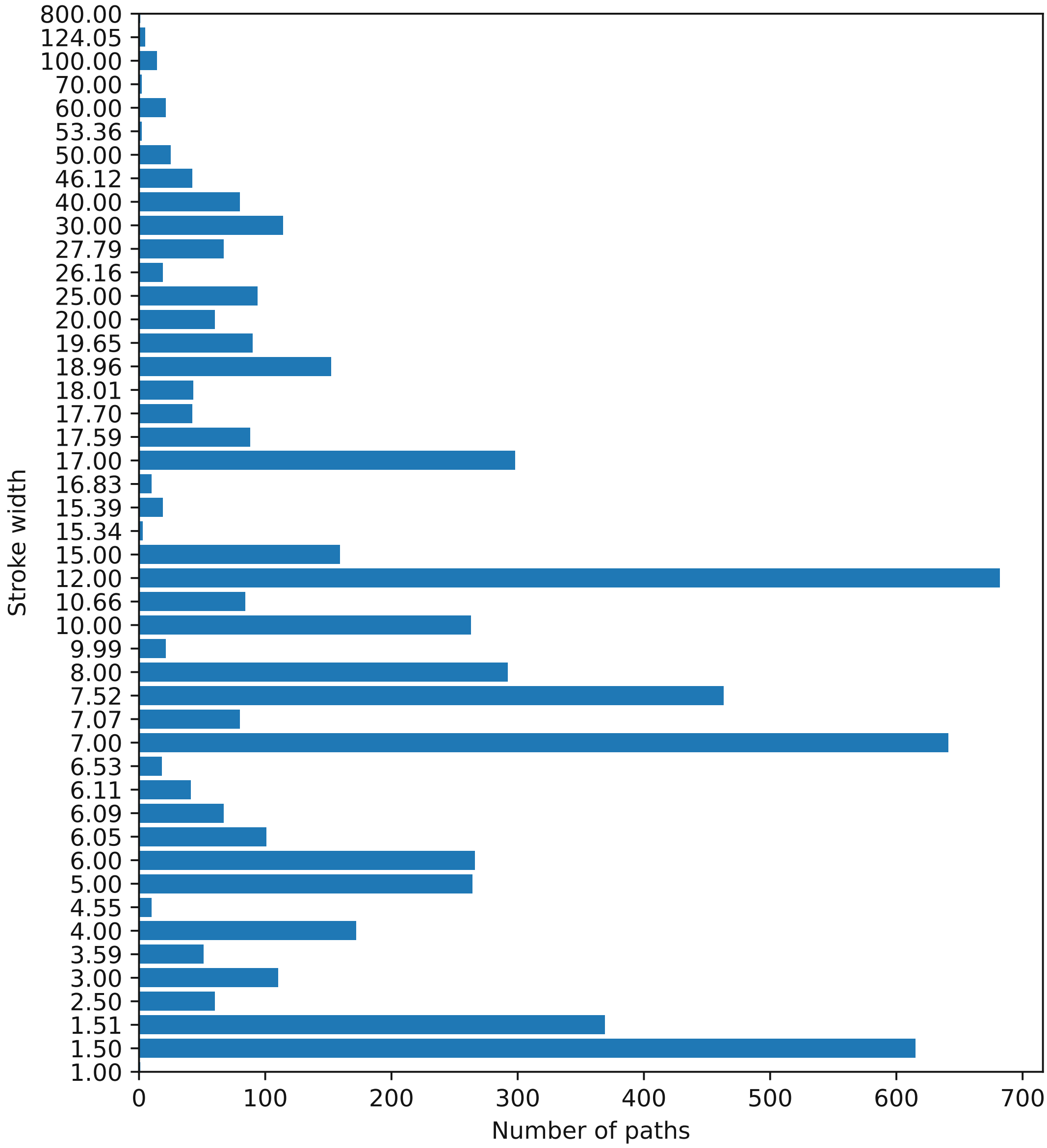


Figure 4.16: Number of curves per stroke width other than 2px in the Tonari clean animation frames.

hex string	Color	Part of schema	Number of paths
#000000		True	45289
#0000ff		True	13944
#ffffff		False	5149
#ff0000		True	3728
#00ff00		True	486
#120a0a		False	76
#0c0606		False	63
#180d0d		False	10
#ff8000		False	7
#49ff49		False	6

Table 4.5: The colors used in the Tonari clean animation frames. Colors not part of the clean animation frame schema defined in Figure 2.10 are indicated.

(a) The clean animation frame with the irregular patch.

(b) The clean animation frame with the irregular patch removed.

Figure 4.17: An example of an irregular patch in an animation sequence frame provided by Tonari animation.

The other irregularity is the existence paths with a stroke width that differs from the constant stroke width defined for the image. These are present in some Tonari clean animation frames. Keeping in mind that the constant stroke width for clean animation frames is set to 2px, the distribution of paths with a different stroke width is displayed in Figure 4.16. They are mostly either white patches as described in Section 4.2.3 and Figure 4.13 or spurious indications and corrections in the sample animation sequence frames displayed in Figure 4.17.

While it might seem trivial to manually fix the irregularities mentioned in this section at a first glance, it turns out to be impossible for non-professional artists. Instead, automatic preprocessing steps are taken to reduce the amount of irregular curves, which are described in Section 4.2.4. However, note that the preprocessing steps are rudimentary and conservative, and do not exhaustively remove all irregular curves. This is especially the case for the types of overlapping curves depicted in Figure 4.15, where it is impossible to unambiguously remove one of the overlapping curves.

4.2.4 Processing

The dataset described in this section consists of human-generated and synthetic vector images, where the former are stored on disk in the SVG format. Before these human-generated images can be used to train or evaluate the method, several preprocessing steps have to be applied, which are described below. The images are processed in parallel using GNU parallel 20220922 [Tange, 2022], with the processing steps implemented using Python 3.8.12 [Python Core Team, 2019].

Curve correction

The first preprocessing steps are motivated by the statistics and irregularities mentioned in Table 4.4 and Section 4.2.3. Since the line-art vectorization method is based on curves instead of paths as graphical primitive, the vector images are converted into flat images only consisting of curves. In order words, paths and other group structures are removed from the vector files using Inkscape 1.0 [Inkscape Project, 2020].

The next preprocessing step is to remove irregular curves from the Tonari subset. These include curves with a color that is not allowed by the clean animation frame schema defined in Figure 2.10 and curves with a stroke width that is different from the constant stroke width defined for the clean animation frames (i.e., 2px). As already established in Table 4.5, the incorrectly colored curves are few in number, thus making them safe to remove. The curves with irregular stroke width are larger in number, but are not part of the semantic structure of the clean animation frame. Recall that they mostly include white patch corrections or other indications as shown in Figures 4.13 and 4.17. Hence, it is safe to remove these curves.

While it might seem visually counter-intuitive to remove white patches, as this renders visible curves which the artist intended to hide, it makes sense when considering the underlying vector structure. Drawing white patches over curves only alters their appearance once rasterized, but does not affect their vector representation. Furthermore, these white patches are often drawn over only a part of a curve. Hence, it is challenging to unambiguously derive the correct vector representation of only the curves (or, only the parts of curves) which are intended to be shown. Therefore, white patches are simply removed entirely to lay bare all curves and ensure equivalence between the raster and vector representation of the image. A more conservative alternative would be to sidestep the ambiguity and simply remove all curves which intersect with a white patch. This alternative was not considered in this work as this would greatly reduce the already scarce amount of training data.

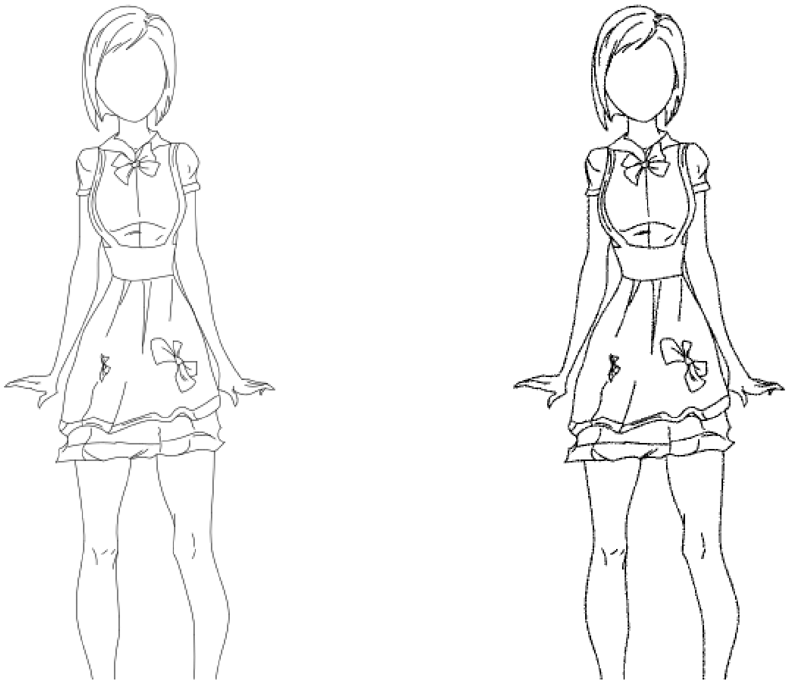
Note that, as white patches make up a large portion of overlapping curves, removing them also reduces this issue. However, there still remain overlapping curves of the type depicted in Figure 4.15, which cannot be easily removed automatically.

For this preprocessing step, the svgelements 1.9.5 Olsen [2022] Python package is used.

Rasterization

It is important that all images in the dataset are available in vector format, which enables them to be used to compute the loss for the model and to evaluate vectorization methods. Furthermore, raster versions of these images are required as input data for the vectorization methods. All subsets in the dataset provide raster image versions of the vector images. However, in order to ensure consistency across the input images, the same rasterization scheme is applied to all.

For the consistent rasterization, one option would be to use the rasterization algorithm used by Clip Studio [CELSYS, Inc., 2021] for all images, since this would perfectly match the clean animation frame images in production at Tonari Animation. However, this was not done for two reasons: firstly, the rasterization algorithm used in Clip Studio could not be determined and cannot easily be used programmatically. Secondly, using a different rasterization algorithm shows that the model does not overfit to a specific rasterization algorithm. Hence, the open source CairoSVG 2.5.2 [CourtBouillon, 2021] algorithm was used for the rasterization. It was applied to all curves of all vector images with consistent settings consisting of a uniform curve color, no fill color and round line endings. The raster images are stored using the PNG format. Furthermore, note that these raster images are not just important as input data, but also to compute the raster-based loss mentioned in Section 4.1.1. The loss is computed using rasterized input images and the model output rasterized using the differentiable rasterizer by Li et al. [2020]. Since the differentiable rasterizer is already used for the model outputs, it would be possible to use the same rasterizer for the inputs as well. However, CairoSVG [CourtBouillon, 2021] is preferred for this purpose due to its significantly lower runtime. Moreover, Figure 4.18 shows that the rasterization output of CairoSVG [CourtBouillon, 2021] is visually similar enough to the differentiable rasterizer [Li et al., 2020].



(a) Rasterization using CairoSVG [CourtBouillon, 2021].

(b) Rasterization using the differentiable rasterizer by Li et al. [2020].

Figure 4.18: The example vector image of Figure 4.9 rasterized using the differentiable rasterizer by Li et al. [2020] and CairoSVG CourtBouillon [2021] with the same settings. Note that, even though the former is a differentiable rasterizer, the results are visually similar, save for slightly thicker strokes.

An important decision of the rasterization algorithm is the resolution. While vector images can be displayed at any resolution, the resolution of the raster images significantly affects the output quality. Since the Tonari clean animation frames were drawn for a 720x405px resolution, it can be assumed that real-world clean animation frames on which the vectorization method needs to be applied will be available at a similar resolution. Hence, the resolution of the rasterization resolution is chosen as the highest exponential of 2 smaller than the width of 720px, i.e., 512x288px. Choosing a resolution that is lower than the target resolution both helps with preventing overfitting to the clean animation frame resolution and trains the model to perform well at lower resolutions. Note that the image is squared to show that the model does not overfit to the aspect ratio of clean animation frames. All vector images in the dataset are rasterized consistently to 512x288px, preserving the aspect ratio with the larger of the width or height set to 512.

Another important decision during rasterization is the stroke width, which is usually set using pixel values and therefore visually connected to the resolution. Due to the homogeneous nature of the stroke width of the processed curves, it can be freely set to a consistent value for all vector images. The stroke width is rather arbitrarily set to 0.512px for all images. This value is a bit lower than the stroke width used for the Tonari clean animation frames, which is 2px for a resolution of 720x405px. This ensures that the method does not overfit to the clean animation frame stroke width. Still, this stroke width in combination with the resolution is large enough that the marked-curve reconstruction model could in theory notice all curves.

The raster images are represented using the RGB color model, i.e., using 3 color channels with interval $[0, 255]$. As described in Section 4.1.1, the channels are scaled by 255 to produce channels with interval $[0, 1]$. Since all raster images are monochrome (even the Tonari clean animation frames after the segmentation by color mentioned in Section 4.1.2), it is also possible to use a single channel instead. However, as noted in Section 4.1.1, both variants perform similarly, with the RGB variant allowing easier generalization to different domains and model architectures.

Data Augmentation

After the preprocessing steps, data augmentation is performed on the human-generated dataset. Data augmentation is another technique to synthetically increase the dataset size and consists of applying meaningful transforms to existing data to derive new data. An additional benefit of data augmentation is that it forces the model to be equivariant to these transformations. Usual image processing data augmentation techniques are based on raster images and cannot be used for this work, since the augmented raster image needs to correspond to the original vector image. Hence, data augmentation techniques are derived for and applied on vector images. The resulting new images are then rasterized to produce corresponding transformed vector and raster images. Similar to the synthetic dataset explained in Section 4.2.2, the data augmentation is performed on-line, i.e., concurrently to the training process.

The data augmentation transformations are efficiently implemented as curve manipulations using the svgpathtools 1.4.4 package [Roth, 2021]. They include curve mirroring, rotation, reversion and dropout. All transformations are independently applied with a probability of 50% per iteration. The transformations are displayed in Figure 4.19 and explained in the following: Curve reversion reverses the orientation of the curves, i.e., swaps the order of the curve parameters, which affects only the vector structure and not the visual representation. While curve mirroring flips curves according to the horizontal axis of the image, curve rotation flips curves according to the diagonal of the image. These augmentations make the model more robust to different curve orientations. On the other hand, curve dropout removes a random percentage of curves lying between $[0\%, 90\%]$ from the image, which is a very important simulation of the input to the marked-curve reconstruction model at various timesteps of the iterative curve reconstruction model. It increases the robustness of the model to partially reconstructed images. Another augmentation technique that was considered is translation, which is obviated by the centering of the input image to the mark location.

(a) Curve mirroring. (b) Curve rotation. (c) Curve reversion. (d) Curve dropout.

Figure 4.19: Various data augmentation transformations applied to the example image displayed in Figure 4.8.

4.3 Evaluation

To answer the RQ1, this section evaluates 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. Prior works include the traditional AutoTrace algorithm [Weber, 2002], the algorithm by Puhachov et al. [2021] combining deep learning and heuristic optimization and two deep learning-based algorithms by Egiazarian et al. [2020], Mo et al. [2021]. The evaluation is performed both qualitatively and quantitatively on a held-out portion of the dataset detailed in Section 4.2. Great care is taken to ensure reproducibility of the evaluation results. For this purpose, Section 4.3.1 extensively describes the experimental setup. Section 4.3.2 lists limitations of the experimental setup. Section 4.3.3 defines the metrics used for the quantitative evaluation. The evaluation results are shown in Sections 4.3.4 and 4.3.5.

4.3.1 Setup

For the experimental setup, the dataset detailed in Section 4.2 is split into a training, validation and test set according to standard practices. While the training split is used to train the model, the validation split is used to evaluate different configurations during the design of the method and the test split is held out to be used for the final evaluation to measure overall performance. Table 4.6 gives a summary of these dataset splits.

Note that in Table 4.6, only Tonari clean animation frames are considered for the test split, since the vectorization of clean animation frames is the objective of this work. Hence, 10 random images from the Tonari subset are used for the test dataset. The remaining dataset is split into a training set and a validation set, where the validation split consists of a random sample of 10% of the high-quality human-generated subsets of the dataset.

Since the test dataset used for the final evaluation is not publicly available, the results

Split	Tonari	SketchBench	TU Berlin	Synthetic	Total
Train	116	383	20000	4100	24599
Validation	13	42			55
Test	10				10

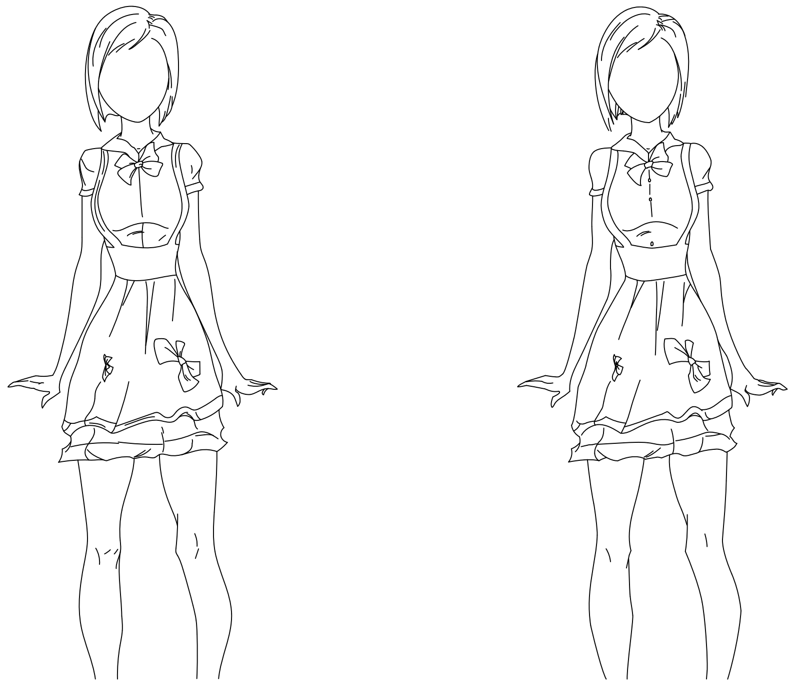
Table 4.6: Distribution of the dataset splits over the dataset subsets displayed in Table 4.3. Note that the synthetic subset is newly generated for each epoch.

are tricky to reproduce. In order to increase the reproducibility of results, a separate evaluation is conducted on a publicly available dataset. For this, a random selection of 41 images of the SketchBench subset is used. In order to attempt to approximate the results on the Tonari subset, these images are drawn from the freeform genre of the dataset, since they resemble clean animation frames the most. Care is taken to achieve an even distribution of artists. Furthermore, every version of the same reference image is included to prevent information leak into the training set. This can be seen in Figure 4.20. Importantly, this separate evaluation includes model training with all SketchBench test dataset images removed from the training dataset, in order to prevent information leak from the test dataset. Note that the Tonari subset remains a part of the training dataset, making the training challenging to reproduce. Therefore, the trained model is provided at `https://github.com/nopperl/marked-lineart-vectorization`.

Implementation

The experiments were conducted on a CentOS Linux release 7.9.2009 machine equipped with an NVIDIA GeForce GTX 2080 Ti GPU (with 11 gigabyte of dedicated memory) using the NVIDIA driver version 460.32.03. The algorithm is implemented in Python 3.8.12 [Python Core Team, 2019] with the PyTorch 1.8.1 [Paszke et al., 2019] deep learning framework using Compute Unified Device Architecture (CUDA) 10.2 [NVIDIA Corporation, 2020] and CUDA Deep Neural Network (cuDNN) 7.0 [Chetlur et al., 2014] for GPU acceleration. Torchvision 0.9.1 [TorchVision maintainers and contributors, 2016], Pillow 8.4 [Clark, 2015], scikit-image 0.18.1 [van der Walt et al., 2014] and ImageMagick 7.0.10 [ImageMagick Studio LLC, 2023] are used for efficient image processing. Throughout the implementation of this work, the number 1234 is consistently used to seed random number generators.

Care was taken to optimize the algorithm and evaluation scripts for computational efficiency. The marked-curve reconstruction model is implemented using PyTorch routines which are compatible with the Open Neural Network Exchange (ONNX) standard [Bai et al., 2019]. This enables the PyTorch model to be converted to an ONNX model. This model format is optimized for inference and provides a wide range of runtimes for various computing environments. The ONNX model is deployed using ONNX Runtime 1.15.1 [ONNX Runtime developers, 2021] in order to vectorize the test set images for the evaluation results. Note that the CUDA execution provider is used for this purpose, as initial experiments showed it to be faster than the NVIDIA TensorRT or the central



(a) Drawn by Branislav Mirkovic [Yan et al., 2020].
(b) Drawn by Maria Hegedus [Yan et al., 2020].

Figure 4.20: Two clean line-art vector images drawn by different artists based on the rough reference sketch shown in Figure 4.9. Notice minor difference between the two versions.

processing unit (CPU) execution provider. Table 4.7 shows that inference with the ONNX model is significantly faster than with the PyTorch model.

Recall that the marked-curve reconstruction model consists of roughly 2M learnable parameters, as described in Section 4.1.1. These model parameters are represented using matrices of 32-bit floating point numbers [Institute of Electrical and Electronics Engineers, 2019]. This is the default datatype of PyTorch and was assumed to be precise enough. Furthermore, since the amount of parameters is small enough for the available GPU memory, data types with a lower number of bits per number or automatic mixed precision (AMP) [Micikevicius et al., 2018] are not considered.

Since an important aspect of this work is reproducibility, the code repository is open sourced at `https://github.com/nopperl/marked-lineart-vectorization`. It includes code to train and evaluate the model as well as code to reproduce statistics, graphs and tables displayed in this work. Furthermore, it is accompanied with an extensive readme on how to set up and reproduce the experiments, figures and tables in

	torch		ONNX	
	median	IQR	median	IQR
subset				
tonari	26.06	38.76	9.49	13.18
sketchbench	46.93	36.16	15.83	16.14

Table 4.7: Comparison of the average runtime measured in seconds of vectorizing an image in the Tonari and the SketchBench test set using the the ONNX and the PyTorch [Paszke et al., 2019] respectively. It can be seen that the faster performance of the ONNX model is statistically significant.

this work.

In order to ensure a reproducible setup, the training and evaluation is run inside a Docker container [Merkel, 2014]. The image for this container can be reproduced by building it according to the Dockerfile definition provided in the open source repository. The base image used is `pytorch/pytorch:1.8.1-cuda10.2-cudnn7-devel` and is available in the Docker Hub. As an alternative, an installation script for creating an Anaconda [Anaconda, 2020] environment with all required dependencies is provided.

Further tools used for the evaluation include pandas 2.0.3 [McKinney, 2010] for data manipulation, SciPy 1.11.1 [Virtanen et al., 2020] for statistics, Matplotlib 3.5.0 [Hunter, 2007] for plots and NumPy 1.21.4 [Harris et al., 2020] for CPU-based numerical computation.

Prior work

In addition to the line-art vectorization algorithm developed in this work, a range of prior work is evaluated on the test datasets. These consist of state-of-the-art methods with publicly available code, which is important for reproducibility. In detail, prior works considered for the evaluation are:

- AutoTrace [Weber, 2002], which is a widely used traditional line-art image vectorization method, which fits the intended task well since it works on clean line-art images and outputs bezier curves,
- The line-art image vectorization algorithm developed by Puhachov et al. [2021], which combines deep-learning based keypoint extraction with a curve reconstruction algorithm using poly-vectors and geometric flows,
- A deep learning-based algorithm trained using vector supervision by Egiazarian et al. [2020] primarily to vectorize technical line drawings, and
- A deep learning-based line-art vectorization algorithm by Mo et al. [2021] trained solely using raster supervision.

Similar to the case for the marked-curve reconstruction model, Docker containers are used for the inference of models of prior work. However, note that building a single Docker image for both the marked-curve reconstruction model and prior work is impossible due to a range of mutually conflicting dependencies. Hence, for the inference of prior work, separate Dockerfiles are created for reproducibility and contributed to the respective code repositories. Each prior work evaluation is performed in a separate Docker container.

A consistent, identical line-art image vectorization procedure is set up for all methods. The test set line-art raster images are cleaned and segmented by color into grayscale raster iamges. These images are input as-is into the respective methods. The outputs are processed to SVG files of a consistent format with the aspect ratio given by the input image, consisting of cubic bezier curves. Additionally, all output vector images are rasterized using CairoSVG [CourtBouillon, 2021] with white background and at the stroke width returned by the respective method. The vectorization pipeline is completely rerun for each image, i.e., for each image, the algorithm is set up and the model is loaded into memory from scratch, thereby avoiding potential unintended information leak and to ensure consistent runtime measurements.

In general, the raster line-art image vectorization is performed using the default hyperparameters for all prior work. However, some necessary alterations are taken, which are described in the following.

AutoTrace [Weber, 2002] By default, the AutoTrace algorithm [Weber, 2002] traces outer lines of strokes. However, since clean animation frames use a constant stroke width and do not contain holes within a stroke, outerline vectorization yields redundant results and unnecessarily introduces potential errors. Hence, the AutoTrace algorithm is instructed to perform centerline tracing using the `-centerline` argument. Furthermore, the background color for input raster images is defined to be white to improve the curve detection.

Deep Vectorization of Technical Line Drawings [Egiazarian et al., 2020] The line drawing image vectorization algorithm developed by Egiazarian et al. [2020] processes the input raster image into tiles to individually vectorize. During this process, the tiles are repatched by calculating a scale which takes the number of curves identified in the tiles into account. In some cases this scale was rounded to 0, causing a divison by zero. The algorithm was altered to set the minimum of this repatch scale to 1.

Virtual Sketching [Mo et al., 2021] By default, the virtual sketching algorithm introduced by Mo et al. [2021] generates 10 vector images given a single raster image, with each vector image reproduction starting from a different random curve. The user is supposed to choose the best image out of the 10. In order to stay consistent with other works and to decrease the runtime, only one vector image is generated per raster input image. Furthermore, the virtual sketching algorithm produces curves with a dynamic stroke width according to a linear schedule. However, their SVG conversion script converts these stroke widths to a constant value. This parameterization is actually closer to the intended output for this work and therefore maintained.

Polyvector Flow [Puhachov et al., 2021] Puhachov et al. [2021] developed and evaluated their line-art image vectorization method using the proprietary Gurobi 9.1.1 library [Gurobi Optimization, LLC, 2023]. An academic license could be acquired for this library, but was incompatible with version 9.1.1. Hence, version 9.1.2 is used for the evaluation. Furthermore, as described in Chapter 3, the algorithm uses the `polyline` SVG primitive to represent strokes, which is overparamterized for the images considered in this work. This primitive is converted to a sequence of cubic bezier curves to stay consistent with the ground truth test dataset and other methods. Just like other methods, the comparison with the test data is performed at an individual curve level, thereby discarding the sequence information. An alternative would be to compare the output vector images with the ground truth vector images at a cubic bezier curve *sequence* level, since the ground truth also includes cubic bezier curve sequences stored using the `path` SVG primitive. However, in order to stay consistent with other methods and due to the low number of curves per path in the ground truth (see Section 4.2.1), this is not done.

4.3.2 Limitations

It is important to keep in mind that this evaluation of the performance of clean line-art image vectorization algorithms is limited by a range of factors. The most important limiting factor is the data. The reproducibility of results is significantly affected by the proprietary nature of the test dataset used for the evaluation. In order to alleviate this issue and provide reproducible results, in addition to the Tonari clean animation frames, a random subset of the SketchBench [Yan et al., 2020] clean line-art images is used as additional test dataset for evaluation. Furthermore, note that while the marked-curve reconstruction model was not trained on the Tonari test set, it was trained on data coming from the same domain of clean animation frames. This is not the the case for other methods, which were mostly trained on data from adjacent domains such as professional sketches.

A further limiting factor is the experimental setup. As described in Section 4.3.1, each method is run in its own Docker container. Hence, the execution environment is different for all methods. The influence of the execution environment on the evaluation results could not be determined.

Moreover, there are limitations that are specific to each prior work. For one, the deep line drawing vectorization algorithm proposed by Egiazarian et al. [2020] includes a physics-based algorithm which refines the curves reconstructed by the trained Transformer [Vaswani et al., 2017] model, as described in Section 3.1. Figure 3.2 shows that it significantly improves their result and can be considered integral to their method. However, it is important to keep in mind that it is not differentiable. Furthermore, the same algorithm could be applied to other methods to improve their results. This is not done, thereby lending a theoretical advantage to the method by Egiazarian et al. [2020].

Another limitation specific to AutoTrace [Weber, 2002] is that it is the only method not run on a GPU. This is done since a GPU implementation of the algorithm could not be found and it is unclear whether it can be efficiently implemented using General-purpose computing on graphics processing units (GPGPU) in the first place. This limitation could theoretically affect the runtime of the algorithm.

Furthermore, there are two limitations associated with the line-art vectorization algorithm developed by Puhachov et al. [2021]. As described in Section 4.3.1, it depends on the proprietary Gurobi library [Gurobi Optimization, LLC, 2023], which diminishes the reproducibility of their results. Furthermore, note that the algorithm segfaults when the number of black pixels is very low in an image. This happened for the lime color segment of one image at 512px resolution in the Tonari test dataset, which is therefore not considered in the evaluation.

4.3.3 Metrics

In order to quantify the extent to which the line-art vectorization method developed in this work and related state-of-the-art methods are able to automatically vectorize clean animation frames, consistent metrics are used to calculate the difference between the ground truth (i.e., the gold standard) and the vectorization results. This section explains the metrics used for this purpose.

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 bezier curves of arbitrary length n and each cubic bezier 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 as explained in Sections 2.2.1 and 4.1). The metrics measure how well $\hat{\mathbf{Y}}$ matches the ground truth vector image $\mathbf{Y}$ corresponding to the ground truth input image $\mathbf{X}_{\text{raster}}$, where again $\mathbf{Y} = (\mathbf{y}_j)_{j=0}^{m}$ is a sequence of cubic bezier curves of length m.

IoU Based on the RQ1, an important characteristic of the output vector image is its visual similarity to the ground truth. Following related works [Egiazarian et al., 2020, Mo et al., 2021, Guo et al., 2019], this is measured using the IoU metric defined in Equation (2.10). Recall that the IoU ranges from $[0, 1]$, where 1 indicates a perfect match and 0 a perfect miss. To calculate the confusion matrix for Equation (2.10) (see Table 2.1), the rasterized output image $\hat{\mathbf{Y}}_{\text{raster}}$ and the raster input image $\mathbf{X}_{\text{raster}}$ are binarized, with black pixels defined as true values and white pixels defined as false values. Since the IoU is calculated using raster images, it follows that it measures only how well the output bitmap overlays the input bitmap [Yan et al., 2020, Puhachov et al., 2021] and does not consider the vector structure when comparing the images.

Curve error The second important characteristic when answering the RQ1 is that the vector structure and primitives of the output image $\hat{\mathbf{Y}}$ match the ground truth image $\mathbf{Y}$. As explained in Sections 2.1 and 2.1.3, this is tricky to measure, as there are multiple semantically correct vector images that can be produced given a raster image as input. For the evaluation, the structure of the ground truth images in the test dataset is considered the gold standard, and vector outputs are evaluated by how close their structure is to the ground truth.

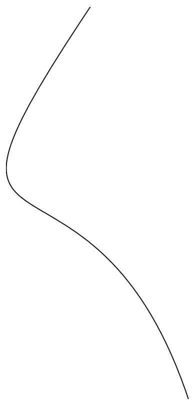
Since the work discards hierarchical information of the vector image and represents vector images as a single sequence of cubic bezier curves (see Section 4.2.1), the vector structure of images is measured at the individual cubic bezier curve level. In order to measure how close the cubic bezier curves of the output image $\hat{\mathbf{Y}} = (\hat{\mathbf{y}}_j)_{j=0}^n$ are to the ones in the ground truth image $\mathbf{Y} = (\mathbf{y}_j)_{j=0}^m$, the minimum error to the ground truth curves $\min_{j=0}^m d(\hat{\mathbf{y}}, \mathbf{y}_j)$ is calculated for each output curve $\hat{\mathbf{y}}$. As there is no natural relation between output and input curves, the curves with the minimal distance to each other are assumed to match. The total curve error is then defined by Equation (4.2) as the average of these individual curve errors, with lower values indicating a better fit to the ground truth vector structure.

$$\text{curve error}(\hat{\mathbf{Y}}, \mathbf{Y}) = \mu_{i=0}^n \left(\min_{j=0}^m d(\hat{\mathbf{y}}_i, \mathbf{y}_j) \right) \tag{4.2}$$

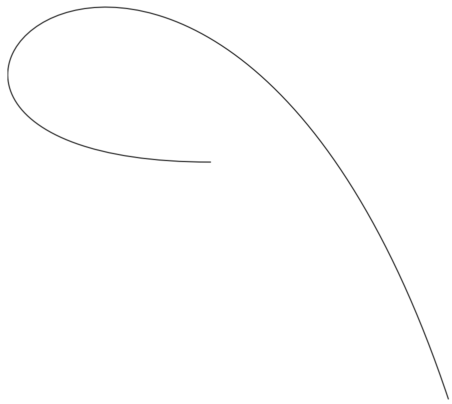
In detail, the function chosen to measure the individual curve error $d(\hat{\mathbf{y}}, \mathbf{y})$ is the sum of absolute errors defined in Equation (4.3). It calculates the sum of the absolute difference of each curve parameter $|\hat{y}_i - y_i|$. There are two considerations for why this function is chosen. Firstly, it is important to calculate the sum of curve parameter errors instead of the mean, as is done in the MAE defined by Equation (2.8). This leads to the error scaling linearly with errors in individual parameters $|\hat{y}_i - y_i|$. In the MAE, the contribution of individual parameter errors is diminished by taking the mean of all errors. As an example, consider the two cubic bezier curves shown in Figure 4.21, in which all parameters match except a single number. Even though nearly all parameters are identical, their visual representation is completely different. This is not well reflected in the MAE, which is 0.375 for the example. On the other hand, the sum of absolute errors is 3, which is exactly the absolute difference of the deviating number. A disadvantage of using the sum instead of the mean is that it does not normalize for the sequence length, leading to insufficient comparability of two sequences of varying length. However, since all curves compared using this metric consist of 4 points (i.e., 8 numbers), this is of no further concern.

$$d(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=0}^{8} |\hat{y}_i - y_i| \tag{4.3}$$

The second consideration for the error function chosen is that the absolute error is used instead of the squared error. This is due to the fact that the squared error diminishes small differences and exaggerates large differences, as described in Section 2.3.2. Furthermore, the absolute value is more efficient to compute than the squared value.



(a) A cubic bezier curve parameterized with the start point $(4, 5)$, end point $(3, 1)$ and control points $(3, 2)$ and $(1, 4)$.



(b) A cubic bezier curve parameterized with the start point $(4, 5)$, end point $(3, 4)$ and control points $(3, 2)$ and $(1, 4)$.

Figure 4.21: Two cubic bezier curves with identical parameters except the y-coordinate of the end point. Note, that even though nearly all parameters are identical, the visual representation of the curve is completely different.

As an aside, using the IoU as error function $d(\hat{\mathbf{y}}, \mathbf{y}_j)$ for the total curve error was considered as an additional metric. However, this metric was ultimately not implemented due to high computational requirements.

Curve ratio Recall that the output vector image $\hat{\mathbf{Y}} = (\hat{\mathbf{y}}_j)_{j=0}^{n}$ and the ground truth vector image $\mathbf{Y} = (\mathbf{y}_j)_{j=0}^{m}$ consist of n and m cubic bezier curves, respectively. 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 $n/m = 1$. In the case of mismatching vector structures, $n/m \neq 1$, with values closer to 1 indicating closer matches. Note that this metric should be considered in combination with the IoU, since it is also possible for the number of curves to match for visually dissimilar images.

Curve length The average curve length is an interesting property of vectorization methods, as it shows the kind of primitives the methods are biased towards. The value is calculated by evaluating the cubic bezier curves and measured in pixels. Note that there is no semantic preference towards shorter or longer curves. Hence, it makes sense to consider this information in combination with the average curve length in the ground truth listed in Table 4.8. Values closer to the ground truth can be considered as representing a closer match to the ground truth vector structure. However, the curve length alone is neither sufficient nor necessary to conclude that.

Curve distance An important quality of a line-art vector image is that there are no unintended holes between curves. As explained in Section 2.2.1, unintended holes in clean animation frames increase the difficulty of successive steps in the limited animation process. Furthermore, since junctions can only be represented using coincident curves, having holes in intended junctions can have an adverse effect on downstream applications [Yan et al., 2020]. Therefore, following Yan et al. [2020], holes between curves are measured using the the minimum distance of each curve endpoints to each other curve endpoints. Similar to Yan et al. [2020], the minimum distances are aggregated by sum instead of the average, since the average would benefit methods that erroneously output short curves with small distances for visually continuous curves in the input image. In detail, given an output vector image $\hat{\mathbf{Y}} = (\hat{\mathbf{y}}_j)_{j=0}^{n}$, the metric is defined by Equation (4.4), where $\mathcal{E} = [0, 1, 6, 7]$ defines the indices of the start and the end point parameters of a curve. There exists a small number of outliers with a minimum distance larger than 50 pixels, which are ignored since they are too far removed from other curves to be considered constituting holes within curve sequences.

$$\mu_{i=0}^{n}\left(\min_{j=0}^{n}\sum_{k\in\mathcal{E}}|\hat{y}_k^i - \hat{y}_k^j|\right) \tag{4.4}$$

Note that there are two limitations associated with this metric. Firstly, it unduly considers some types of curves as having holes. Figure 4.22 shows that there exist curves which are intentionally not connected to any other curve, but are placed in the close vicinity of other curves. Secondly, the metric does not normalize for the sequence length, which limits comparability since most methods output a different numbers of curves for the same raster image. Hence, as with the curve-length metric, the curve distance has to be considered together with the corresponding ground-truth metric in Table 4.8. 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. For values that are smaller than the ground-truth baseline, two interpretations can be considered: If the curve ratio $n/m < 1$, it follows that the output vector image simply contains fewer holes in proportion to the curve ratio. Otherwise, it shows that the method outputs fewer unintentional curves than the ground truth, which is unlikely but not impossible, due to the irregularities described in Section 4.2.3.

Runtime A metric only tangentially related to the correct vectorization of a raster image, but still important for the application of the vectorization method is the runtime required to produce a vector image. The metric is measured in seconds, with lower values indicating faster algorithms. The runtime is measured using GNU Time [Keppel et al., 2018] by adding the CPU time spent in the kernel and in the user mode. An alternative would be to take the wall clock time, i.e., the total elapsed time between start and finish of the process. However, this value is not used since it could be inconsistently influenced by other processes blocking required resources.

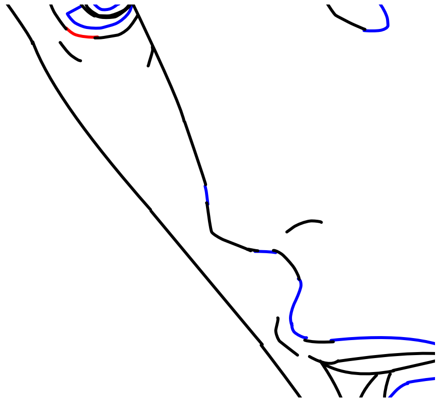


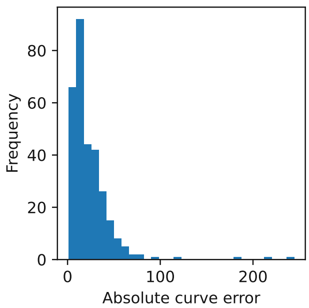
Figure 4.22: The clean animation frame shown in Figure 2.11 zoomed into the nose region, which contains intentionally unconnected curves. Such curves include the black curve representing the right nostril, the black curve representing the upper lip or the black curve representing the crinkle below the left eye.

GPU Memory usage Another performance metric only tangentially related to the correct vectorization of a raster image is the maximum amount of dedicated GPU memory required by the process. This is important since dedicated GPU memory is often a limiting factor. It is measured using the `nvidia-smi` tool in mebibyte (MiB), where lower values indicate less memory usage. It is not measured for AutoTrace [Weber, 2002] since it does not use the GPU.

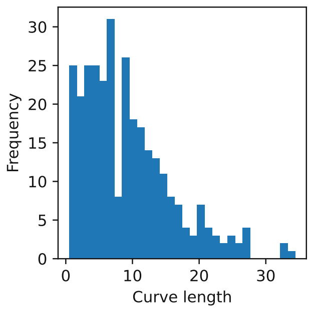
Another important characteristic of output vector images is the number of overlapping curves, which could be calculated similar to Section 4.2.1. However, since the calculation of overlapping curves is computationally expensive, this metric was not considered. Furthermore, overlapping curves are less of an issue than holes between curves, as described in Section 1.3.

Each performance metric is calculated per curve, leading to a metric distribution over all curves for each image in the test dataset. The two exceptions to this are the IoU and the runtime, which are calculated per image, leading to a distribution over all images in the test dataset. In any case, these distributions are represented using aggregate measures for location and skew. Usually, the mean and standard deviation are used for this purpose. However, these measures have low statistical efficiency when applied to non-normal data. Figure 4.23 shows the distributions for the metrics of the line-art vectorization method developed in this work applied to the image shown in Figure 4.1. It can be seen that the distributions do not follow the normal distribution, as they are left-skewed and contain outliers. This is corroborated by applying the Shapiro-Wilk test [Shapiro and Wilk, 1965] on the data, which yields a p-value lower than 0.05 for all distributions, thereby rejecting the null hypothesis that the samples were drawn from the normal distribution. Thus, the median and the IQR are chosen as robust alternatives for the location and skew measure. The only exception is the IoU shown in Figure 4.23d, which is reasonably normal distributed. Still, to keep metrics consistent, the median and IQR are used as aggregate measures for the IoU as well.

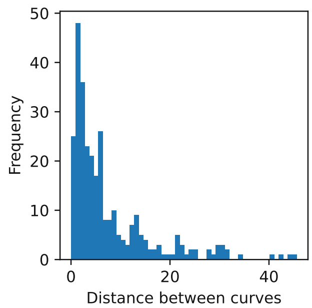
The metric distributions and other distributions are often compared against each other in subsequent sections, e.g. when comparing differences in model performance. In order to ascertain whether differences are actually statistically significant, the Wilcoxon-Mann-Whitney U-Test [Wilcoxon, 1945, Mann and Whitney, 1947] is used. This test is used because it is nonparametric and has relaxed assumptions of the underlying distribution. The null hypothesis H_0 is that the model metrics follow the same distribution. The alternative hypothesis H_A is that they follow different distributions. The primary interest lies in the *location* of the distributions. The output of the statistical test is a p-value, which indicates the probability of the observed sample given the assumption of H_0 being true. Under the assumption of a significance level of $\alpha = 95\% = 0.95$, if $p < 1 - \alpha = 0.05$, H_0 is rejected and H_A is accepted. This setup is used throughout the text when statistical significance is mentioned.



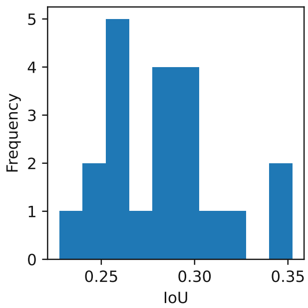
(a) Distribution of the curve error.



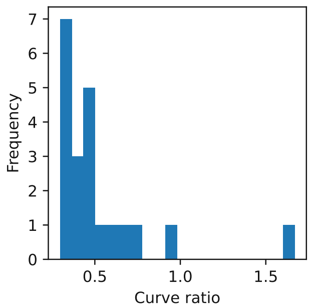
(b) Distribution of the curve length.



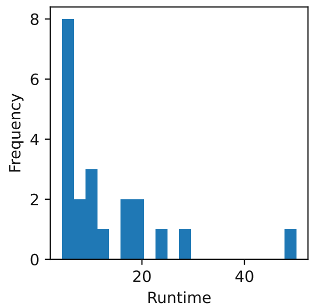
(c) Distribution of the minimum curve distance.



(d) Distribution of the IoU for all images in the Tonari test set.



(e) Distribution of the curve ratio for all images in the Tonari test set.



(f) Distribution of the runtime for all images in the Tonari test set.

Figure 4.23: The distributions of all metrics calculated for the line-art image vectorization method developed in this work. The top three metrics are calculated for the image shown in Figure 4.1. The bottom three metrics are calculated for the entire Tonari test dataset.

		curves		curve length		curve distance	
		median	IQR	median	IQR	median	IQR
tonari	512-0.512	205.00	423.00	2.56	0.79	1555.82	1832.41
	1024-1.024	205.00	423.00	5.12	1.57	1725.81	3489.33
sketchbench	512-0.512	208.00	266.00	12.43	14.29	2355.22	1237.41
	1024-1.024	208.00	266.00	24.85	28.57	2726.03	2330.49

Table 4.8: Selected metrics of the vector images in the test dataset. This information can be used as baseline for the corresponding metrics in Table 4.9. Note that the ground truth images are scaled to all evaluation resolutions to produce baseline values in all resolutions for convenience.

4.3.4 Quantitative Evaluation

To answer the RQ1, this section provides a quantitative evaluation of the extent to which the line-art vectorization method developed in this work and related state-of-the-art methods are able to automatically vectorize clean animation frames. For this purpose, the methods are applied to vectorize a test dataset consisting of these images. Using the metrics described in Section 4.3.3, it is possible to ascertain and compare the results of the methods.

		autotrace	polyvector-flow	virtual-sketching	deepvec-techdraw	marked
IoU ↑	median	0.02	0.12	*0.29*	0.28	**0.30**
	IQR	0.01	0.11	0.06	0.07	0.05
curve ratio	median	0.23	1.35	0.30	0.19	0.43
	IQR	0.11	1.08	0.14	0.13	0.15
curve length	median	1.00	0.55	11.16	9.06	8.19
	IQR	0.41	0.04	4.18	2.10	2.45
curve distance	median	891.00	439.18	1442.91	917.50	1361.28
	IQR	1535.00	1126.02	1782.80	974.06	1498.03
curve error ↓	median	20.37	**14.05**	20.08	17.58	*16.76*
	IQR	19.15	14.64	9.12	10.03	5.36
runtime ↓	median	**0.35**	14.82	22.99	97.73	*9.49*
	IQR	0.31	23.95	7.34	72.20	13.18

Table 4.9: 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.

Table 4.9 shows the performance of the following line-art image vectorization methods:

		autotrace	polyvector-flow	virtual-sketching	deepvec-techdraw	marked
IoU ↑	median	0.02	0.09	*0.27*	0.27	**0.30**
	IQR	0.01	0.05	0.04	0.05	0.05
curve ratio	median	1.01	3.47	1.00	0.71	1.36
	IQR	0.83	1.35	0.77	0.46	0.72
curve length	median	1.00	0.55	14.27	9.70	11.38
	IQR	0.41	0.03	2.00	2.78	4.01
curve distance	median	2359.00	863.19	2619.79	1533.96	2549.28
	IQR	641.00	1179.56	482.01	387.16	958.53
curve error ↓	median	50.82	**39.73**	51.01	*42.18*	46.72
	IQR	31.58	44.97	29.90	22.00	37.58
runtime ↓	median	**0.43**	25.41	28.00	121.91	*15.83*
	IQR	0.20	24.78	3.87	36.39	16.14

Table 4.10: Comparison of the performance of the marked line-art image vectorization method and four prior works on the SketchBench test subset at a resolution of 512px.

method	memory ↓
polyvector-flow	*1128*
virtual-sketching	2760
deepvectechdraw	1935
marked	**1008**

Table 4.11: Maximum dedicated GPU memory measured in MiB required by the deep learning-based line-art image vectorization methods.

- The method developed in this work, identified as `marked`,
- The traditional algorithm by Weber [2002], identified as `autotrace`,
- The vectorization algorithm combining deep learning and heuristic optimization by Puhachov et al. [2021] identified as `polyvector-flow`,
- The deep learning-based algorithm using raster supervision by Mo et al. [2021], identified as `virtual-sketching`, and
- The deep learning-based algorithm using vector supervision by Egiazarian et al. [2020], identified by `deepvectechdraw`.

The methods are applied on the Tonari clean animation frame test dataset rasterized at a resolution of 512px, while preserving the aspect ratio (see Section 4.2.4). The performance is measured according using metrics explained in Section 4.3.3. Recall that the average curve length and the average curve distance should be close to the ground truth values, which are listed in Table 4.8. The curve ratio is calculated with the number of curves listed in the same table. For the remaining metrics, the results can be interpreted more easily. 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.

Table 4.9 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. [2021] and with a curve distance that is close to the ground truth, just behind the method by Mo et al. [2021]. 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 (see Table 4.11).

Note that the traditional method by Weber [2002] 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. [2021] also achieves a surprisingly low IoU, but also the best curve error. Note, that the output images of this method contain significantly more curves than other methods due to the overparamterezied outputs mentioned in Section 4.3.1. This leads to a low curve distance, since most curves are just splitted parts of a long polyline.

The other two deep learning-based methods by Mo et al. [2021], Egiazarian et al. [2020] 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. [2020] outputs the lowest amounts of curves, but the curves of the method by Mo et al. [2021] are still longer on average, suggesting that this method produces more curves that do not fit the ground truth curves.

Table 4.10 shows that similar results are maintained on the publicly available SketchBench test dataset, demonstrating reproducibility and generalizability of the evaluation. Unsurprisingly, since the SketchBench images in general contain significantly fewer curves than Tonari images (see Section 4.2.1), the curve ratio ends up significantly higher.

In general, recall that the IoU is in $[0, 1]$. Hence, 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.

Results with higher resolution input images

		autotrace	polyvector-flow	virtual-sketching	deepvec-techdraw	marked
IoU ↑	median	0.01	**0.67**	0.49	*0.52*	0.31
	IQR	0.00	0.05	0.04	0.14	0.04
curve ratio	median	0.27	17.49	0.63	0.32	0.39
	IQR	0.14	5.93	0.15	0.13	0.11
curve length	median	1.41	0.50	14.81	17.39	15.75
	IQR	0.41	0.02	2.17	4.80	4.83
curve distance	median	954.00	5068.26	2999.09	1622.45	1745.01
	IQR	2091.00	7893.17	4545.06	1669.95	2070.26
curve error ↓	median	31.99	33.99	*31.52*	**28.81**	34.76
	IQR	25.92	38.28	20.31	12.80	18.17
runtime ↓	median	**1.48**	113.41	38.21	136.90	*9.46*
	IQR	0.25	244.59	27.34	164.06	10.48

Table 4.12: The same comparison as Figure 4.24 with input images of resolution 1024px.

		autotrace	polyvector-flow	virtual-sketching	deepvec-techdraw	marked
IoU ↑	median	0.01	**0.74**	0.50	*0.58*	0.30
	IQR	0.00	0.04	0.03	0.07	0.03
curve ratio	median	1.11	78.40	2.47	0.97	1.31
	IQR	0.55	58.65	1.15	0.46	0.77
curve length	median	1.41	0.52	16.99	26.60	23.40
	IQR	0.00	0.01	1.72	11.05	7.68
curve distance	median	2390.00	12440.78	7069.76	2540.16	2680.15
	IQR	1291.00	5584.74	3125.29	1886.62	1642.49
curve error ↓	median	102.56	109.27	94.82	**72.07**	*93.62*
	IQR	47.00	56.38	55.52	33.63	72.84
runtime ↓	median	**1.84**	280.16	56.01	300.92	*15.73*
	IQR	0.44	132.90	17.81	186.96	13.91

Table 4.13: The same comparison as Table 4.12 on the SketchBench test dataset instead of the Tonari test dataset.

The methods by Weber [2002], Puhachov et al. [2021] performed unusually low on the evaluation in Table 4.9. A potential cause for this was identified as the low resolution of input images at 512px. To ascertain this hypothesis, the evaluation was rerun with input images rasterized at twice the resolution, i.e., 1024px, while preserving the aspect ratio. Note that this does not affect the training of the marked-curve reconstruction model, i.e.

the model trained on a resolution of 512px is used. Keep in mind that this is significantly higher than the standard resolution of clean animation frames considered in this work (see Section 4.2.1). Hence, performance increases of methods at this resolution will likely not materialize when they are applied to real-world clean animation frames, which likely will only be available at a lower resolution.

Table 4.12 shows the evaluation results on higher resolution. Note that the metrics measured in pixels are affected by the increased resolution, i.e., given an identical vector structure, a curve error of 20px at 512px resolution corresponds to 40px at 1024px resolution. It is clear that all prior methods except AutoTrace [Weber, 2002] perform significantly better than at 512px resolution. The method by Puhachov et al. [2021] 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. [2020] 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.

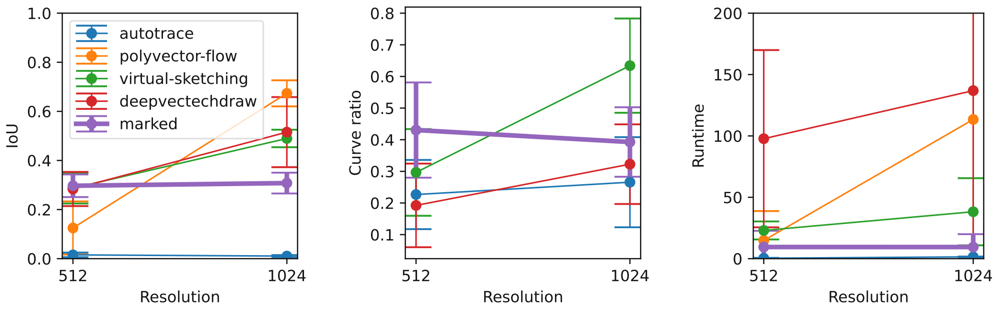
Interestingly, the metrics of the method developed in this work stay remarkably stable at the increased resolution. This intriguing property is investigated in Figure 4.24, which shows the differences of metrics at both resolution sizes. Note that, since metrics measured in pixels scale linearly with the resolution size, they are normalized by the resolution size, i.e., the values of metrics measured at the resolution of 1024px are divided by 2. The method developed in this work is the only one for which metrics do not significantly change depending on the input image resolution. This is especially remarkable for the runtime, which significantly and predictably changes for all other methods. The only exception to this is the curve error and the curve distance, which significantly increases only for the method by Puhachov et al. [2021]. Since the increase in curve distance of this method is especially drastic, it cannot be displayed in Figure 4.24e. Furthermore, the IoU and curve ratio of AutoTrace [Weber, 2002] and the curve length of the method by Egiazarian et al. [2020] do not change significantly. Also of interest is that the curve length of the method by Mo et al. [2021] actually significantly decreases at high resolutions.

Table 4.13 and Figure 4.25 show that similar results are maintained on the SketchBench test dataset, with the curve error of the method developed in this work slightly improving against other methods.

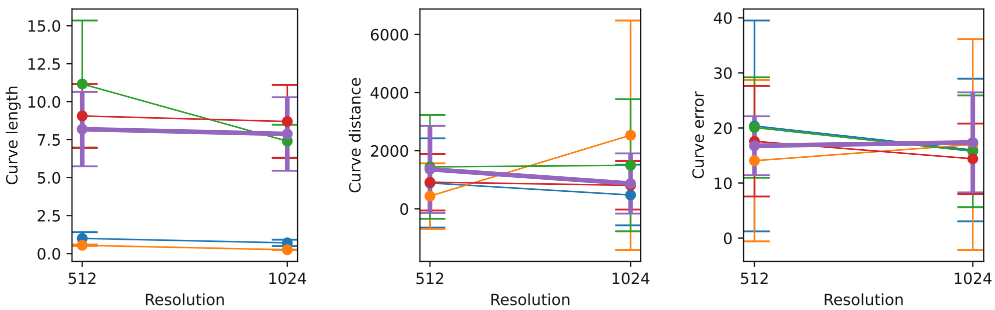
One potential reason for this remarkable input image resolution invariance of the method developed in this work is the selection of reconstruction curves using marks, which explicitly *forces* the model to reconstruct curves which other methods might not have detected. This can be the case for curves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

While other methods achieve significantly better results at the high resolution of 1024px, it needs to be kept in mind that this resolution is higher than the target resolution of clean animation frames by Tonari Animation. Hence, these high results might not materialize when used for real-world data. One potential way to maintain this good performance of other methods with input images at lower resolutions is to apply super-resolution models [Dong et al., 2016] to the input images. However, these models would need to be successfully finetuned to high-resolution line-art images beforehand, which is an open research question on its own.

An interesting question is if this trend continues at even lower resolutions. However, this was not attempted, because resolutions that are significantly lower than 512px are unreasonable small to still expect methods to extract semantically meaningful information out of input images.

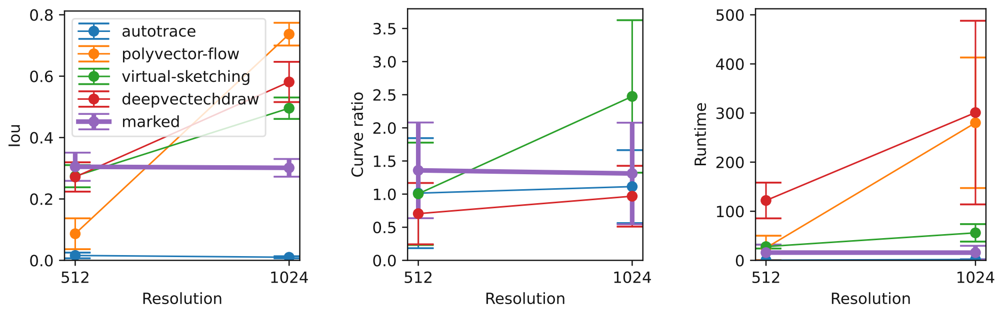


(a) The average IoU. (b) The average curve ratio. (c) The average runtime.

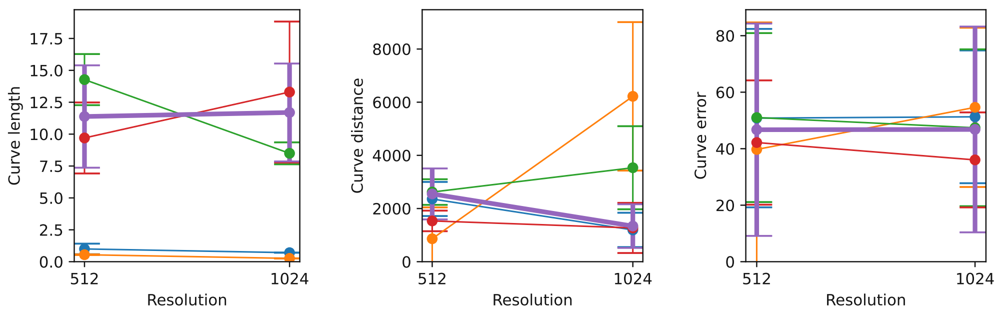


(d) The average curve length. (e) The total curve distance. (f) The average curve error.

Figure 4.24: 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 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.



(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 4.25: The same comparison as in Figure 4.24 on the SketchBench test dataset instead of the Tonari test dataset.

Results with binarized input images

Looking at the results with higher resolution input images, one can assume that the cause for the bad performance of the methods by Weber [2002], Puhachov et al. [2021] in Table 4.9 is that the raster input images are not clearly visible enough. One way to improve this is to binarize the image with a low threshold for black pixels, which accentuates curves. In order to investigate this assumption, the evaluation was rerun using binarized input images. Table 4.14 and Figure 4.26 show that the curve ratio significantly increases for the method by Puhachov et al. [2021] and the curve length significantly increases for both methods, thus corroborating the assumption. This leads to the IoU significantly improving for both methods, with the method by Puhachov et al. [2021] even surpassing the method developed in this work. Furthermore, the method now achieves the lowest curve error, while the improved curve error of AutoTrace [Weber, 2002] is still the worst value of all methods. On the other hand, the curve distance significantly worsens for the method by Puhachov et al. [2021], showing that the low value for non-binarized input images is just an artifact of output images containing a small amount of curves in the first place.

		autotrace	polyvector-flow	virtual-sketching	deepvec-techdraw	marked
IoU ↑	median	0.17	**0.39**	*0.29*	0.18	0.28
	IQR	0.04	0.05	0.02	0.08	0.04
curve ratio	median	0.27	7.32	0.40	0.13	0.44
	IQR	0.11	2.01	0.13	0.09	0.22
curve length	median	9.43	0.50	10.37	9.92	7.99
	IQR	4.39	0.04	3.77	4.87	3.19
curve distance	median	1230.46	2453.15	1723.93	593.07	1413.85
	IQR	1188.55	3520.67	1926.67	498.94	1677.94
curve error ↓	median	19.81	**16.26**	19.77	20.52	*17.68*
	IQR	11.03	9.61	11.32	30.64	5.68
runtime ↓	median	**0.03**	59.22	22.92	79.59	*9.76*
	IQR	0.01	117.67	9.47	33.39	11.42

Table 4.14: The same comparison as Table 4.9 with binarization of images applied prior to running the methods.

However, it can also be seen that binarization is not a silver bullet, as the method by Egiazarian et al. [2020] is actively harmed by it. With binarized input images, it has a significantly worse IoU and a worse curve error.

Interestingly, the changes in curve error are not statistically significant for any method. The methods by Weber [2002], Puhachov et al. [2021] have a significantly higher runtime than with non-binarized input images, further corroborating that they detect and reconstruct more curves given binarized input images. Furthermore, the curve ratio of the method by Mo et al. [2021] significantly increases. Other than that, most metrics do not change significantly.

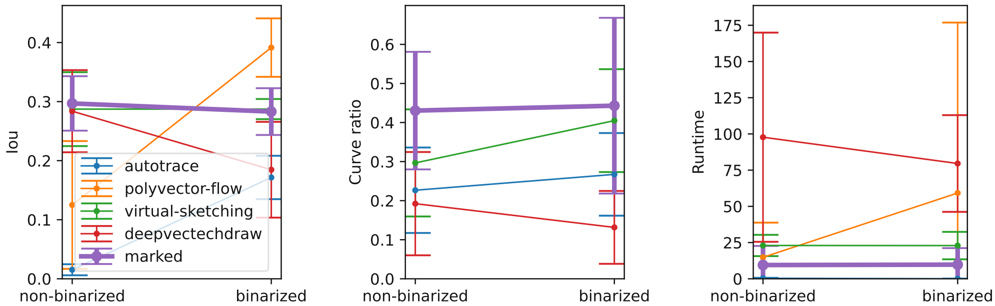
Again, the metrics for the method developed in this work remain remarkably stable on binarized and non-binarized input images, with none changing significantly. Together with the similar result on resolution invariance, this shows that the method developed in this work is more robust to thinner and potentially poorly visible curves, as it is *forced* to reconstruct them when they are marked. Furthermore, the IoU and curve error are only beaten by the method by Puhachov et al. [2021].

		autotrace	polyvector-flow	virtual-sketching	deepvec-techdraw	marked
IoU ↑	median	0.16	**0.41**	0.29	0.18	*0.30*
	IQR	0.03	0.02	0.01	0.07	0.05
curve ratio	median	0.72	33.38	1.63	0.40	1.35
	IQR	0.40	27.34	0.91	0.53	0.77
curve length	median	20.42	0.52	13.89	12.32	11.91
	IQR	8.20	0.02	1.39	2.91	3.24
curve distance	median	1841.21	5900.94	3300.44	1130.48	2512.57
	IQR	883.82	2639.11	1414.49	436.67	875.13
curve error ↓	median	**40.02**	55.18	48.54	57.22	*46.66*
	IQR	12.44	28.78	24.91	24.86	36.51
runtime ↓	median	**0.04**	136.89	27.13	159.14	*16.79*
	IQR	0.00	78.87	5.36	49.33	13.88

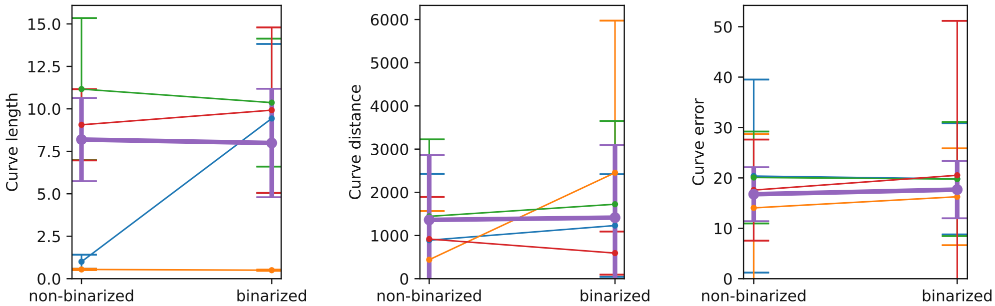
Table 4.15: The same comparison as Table 4.14 on the SketchBench test dataset instead of the Tonari test dataset.

Table 4.15 and Figure 4.27 show that the results hold on the SketchBench test dataset, with the exception of the curve error, which actually significantly worsens for the method by Puhachov et al. [2021].

In general, while the results of the methods by Puhachov et al. [2021], Weber [2002] significantly improve by using binarized input images, this also shows that these methods depend on a high signal-to-noise ratio in the input image. As has been shown, this is easy to achieve for clean line-art images considered in this work by simply binarizing them using a high threshold. However, binarization is non-trivial for images from different domains, which would be the input in a possible cross-domain extension of the line-art vectorization method, as explained in Section 1.3. Under this assumption, methods that are more robust to non-binarized images (and therefore to a lower signal-to-noise ratio) can be considered more suited for a possible extension to cross-domain line-art image vectorization. On the other hand, similar to the super-resolution case mentioned in Section 4.3.4, this can also be solved by training a separate binarization model for the specific domain, which could be applied before using the images as inputs to the methods by Puhachov et al. [2021], Weber [2002].

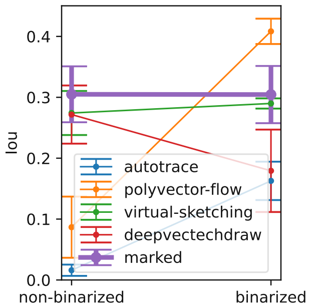


(a) The average IoU. (b) The average curve ratio. (c) The average runtime.

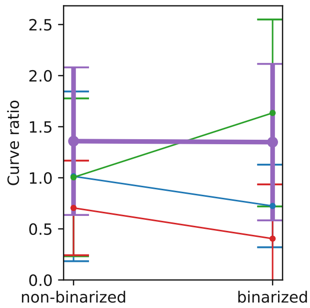


(d) The average curve length. (e) The total curve distance. (f) The average curve error.

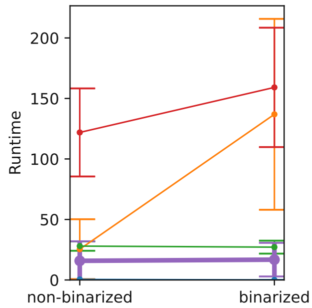
Figure 4.26: Metrics for the line-art image vectorization methods evaluated on binarized and non-binarized images with 512px resolution, respectively. Points denote the median of the metric, while vertical bars denote the IQR. The metrics for the method developed in this work are emphasized. Note that they are not significantly affected by the binarization of images.



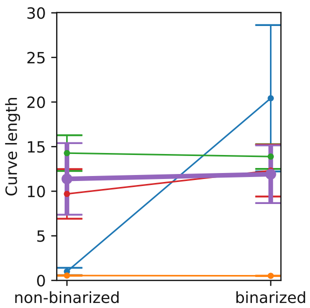
(a) The average IoU.



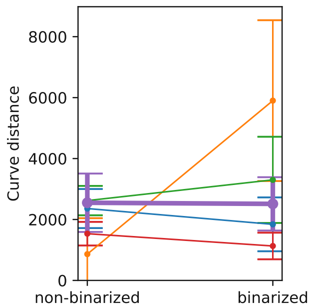
(b) The average curve ratio.



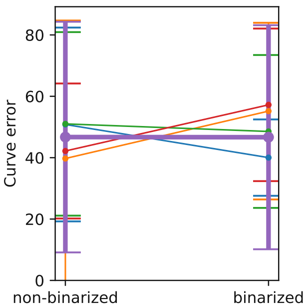
(c) The average runtime.



(d) The average curve length.



(e) The total curve distance.



(f) The average curve error.

Figure 4.27: The same comparison as in Figure 4.26 on the SketchBench test dataset instead of the Tonari test dataset.

Combining higher resolution and binarization

Since it has been established that higher resolution input images and binarized input images significantly improve the results of the methods by Puhachov et al. [2021], Weber [2002] while the method developed in this work remarkably maintains the same performance, a natural investigation is how a combination of higher resolutions and binarization affects the results.

		autotrace	polyvector-flow	virtual-sketching	deepvec-techdraw	marked
IoU ↑	median	0.26	**0.57**	*0.44*	0.34	0.28
	IQR	0.05	0.11	0.02	0.12	0.05
curve ratio	median	0.38	14.33	0.70	0.28	0.40
	IQR	0.22	6.95	0.16	0.16	0.27
curve length	median	18.35	0.56	13.77	17.09	18.67
	IQR	6.26	0.05	3.70	4.19	6.07
curve distance	median	1814.25	4781.61	3242.38	1369.08	1711.39
	IQR	2470.04	8091.26	4562.22	1741.93	2943.68
curve error ↓	median	31.51	**27.89**	*30.60*	33.51	38.82
	IQR	15.64	20.81	16.35	40.54	37.50
runtime ↓	median	**0.05**	110.56	39.43	205.20	*9.34*
	IQR	0.01	246.80	26.99	201.60	11.36

Table 4.16: The same comparison as Table 4.9 with binarized input images of resolution 1024px.

Table 4.16 and Figure 4.28 show the results on binarized high-resolution input images. Note that, again, the curve ratio for the method by Puhachov et al. [2021] is not shown since it is too large in comparison with other methods. Unsurprisingly, it can be seen that all prior methods significantly improve their results. However, some results are significantly worse than with non-binarized high resolution input images. This is the case for the IoU and curve error for the method by Egiazarian et al. [2020]. It also has a lower curve ratio. This corroborates the finding in Section 4.3.4 that binarization of inputs actively harms this method. The case for the IoU also holds for the method by Mo et al. [2021], while the curve error and curve ratio for this method actually significantly improve. The curve error and the curve ratio did not improve significantly with binarization alone. This further shows that binarization is neither completely helpful nor harmful, but has a complex interaction for this method.

Interestingly, the IoU for the method by Puhachov et al. [2021] is also significantly better with non-binarized high-resolution input images than binarized high-resolution input images, even though binarization alone also significantly improved the IoU. On the other hand, the curve error is significantly better than for both non-binarized high-resolution input images and binarized low-resolution input images. Furthermore, the curve ratio

		autotrace	polyvector-flow	virtual-sketching	deepvec-techdraw	marked
IoU ↑	median	0.24	**0.64**	*0.43*	0.36	0.28
	IQR	0.04	0.06	0.01	0.08	0.03
curve ratio	median	1.10	72.27	2.40	1.01	1.35
	IQR	0.47	52.66	1.38	0.86	0.73
curve length	median	30.38	0.58	17.73	19.18	24.24
	IQR	12.68	0.04	1.71	3.21	8.15
curve distance	median	2930.17	11656.21	7108.09	2541.64	2796.84
	IQR	1978.72	5179.87	3309.66	1024.38	1840.35
curve error ↓	median	**69.80**	108.19	*93.70*	97.45	96.15
	IQR	33.87	57.24	55.76	51.75	63.94
runtime ↓	median	**0.07**	313.40	54.40	338.88	*16.99*
	IQR	0.02	204.41	17.86	166.38	11.69

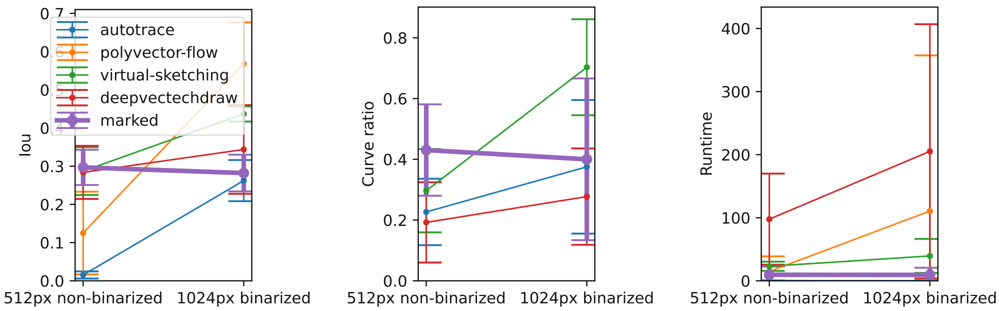
Table 4.17: The same comparison as Table 4.16 on the SketchBench test dataset instead of the Tonari test dataset.

is the highest for non-binarized high resolution input images. This shows that given non-binarized high-resolution input images, the method produces a large number of curves visually fitting the input image but not the ground truth vector structure. Therefore, this suggests that for the output of this method, high-resolution input images are important for the visual quality, while binarization combined with high resolution is important for a semantically correct vector structure.

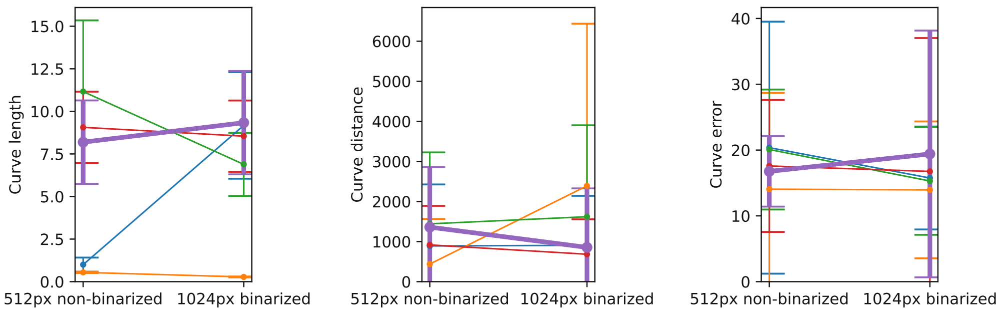
Binarized high-resolution input images seem to be the most helpful for AutoTrace [Weber, 2002]. The IoU, curve error, curve ratio and curve length of AutoTrace [Weber, 2002] significantly improves not only against the non-binarized low-resolution case, but also against the binarized low-resolution case. Recall that Section 4.3.4 showed that high resolutions alone do not help this method at all. This result suggests that high resolutions combined with binarization help the method achieve not only visual resemblance but also matching vector structures.

The metrics for the method developed in this work again stay remarkably stable for all combinations of input image resolutions and binarization. This finally confirms that the method is robust to the input image resolution and binarization. Therefore, it can be concluded that the method can be applied for non-binarized and/or low-resolution input images, which increases its utility. However, note that this comes with the drawback of the results being significantly worse than the best performing method for binarized high-resolution input images.

Table 4.17 and Figure 4.29 show that the results are maintained for the SketchBench test dataset. However, as is the case for binarized input images, the curve error for the method by Puhachov et al. [2021] actually significantly worsens, further showing that the vector structures produced by this method do not match the ground truth for this data domain. Furthermore, the curve error of the method developed by this work slightly improves against other methods, as is the case for non-binarized high-resolution input images (see Section 4.3.4). Finally, the curve distance actually significantly improves for this model, which is the only statistically significant change in all combinations of datasets, image resolutions and binarizations.



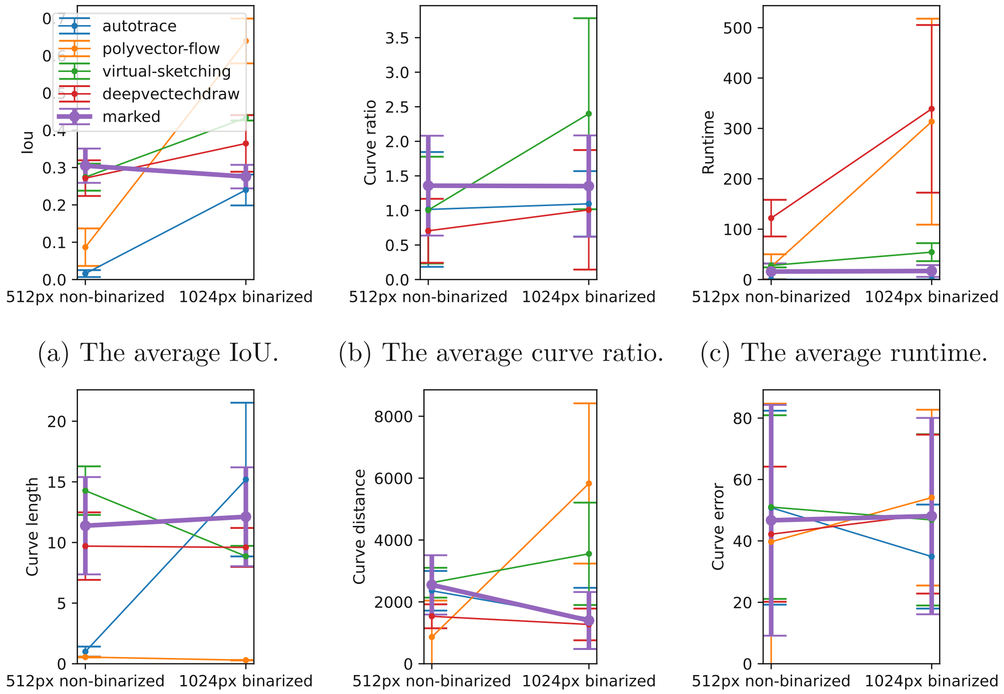
(a) The average IoU. (b) The average curve ratio. (c) The average runtime.



(d) The average curve length. (e) The total curve distance. (f) The average curve error.

Figure 4.28: Metrics for the line-art image vectorization methods evaluated on non-binarized images with 512px resolution and binarized images with 1024px resolution, respectively. Points denote the median of the metric, while vertical bars denote the IQR. The metrics for the method developed in this work are emphasized. Note that they are not significantly affected by the binarization and resolution of images.

For completeness, Figures A.1 to A.4 provide plots for missing combinations of the binarization and resolution of input images. While Figures A.1 and A.2 show how increasing the input image resolution affects results for binarized input images, Figures A.3 and A.4 show how binarization affects results of high resolution input images. Note that on the SketchBench test dataset, there are two metrics that significantly change for the method developed in this work. Firstly, the curve distance is significantly better on high-resolution binarized input images than on low-resolution binarized input images. Secondly, the IoU is significantly worse on binarized high-resolution input images than on non-binarized high-resolution input images. This may suggest that given input images



(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 4.29: The same comparison as in Figure 4.28 on the SketchBench test dataset instead of the Tonari test dataset.

that are both binarized and in high resolution, the model outputs images with worse visual similarity but better semantic correctness of the vector structure. However, since this only concerns a single metric each for one test dataset only, the validity of this conclusion is limited.

4.3.5 Qualitative Evaluation

While the main results of the evaluation are detailed in Section 4.3.4, this section shows some visual results to exemplify the results of the quantitative evaluation.

Figure 4.30 shows the best output of each method for the clean animation frame by Tonari Animation shown in Figure 4.1. The input image has a resolution of 512px. Note that the input image is segmented by color, with each segment vectorized individually as a grayscale image and merged afterwards. Furthermore, the segments are binarized for the methods by Weber [2002], Puhachov et al. [2021], Mo et al. [2021], since that leads to higher-quality outputs.

It can be seen that most methods produce a result that is visually similar to the input raster image, with the exception of the method by Egiazarian et al. [2020]. However, the main objective is to not only achieve visual similarity but also match the semantically correct vector structure of the ground truth vector image associated with the input raster image. Obviously, this is tricky to visualize. Following Guo et al. [2019], Mo et al. [2021], Puhachov et al. [2021], the vector structure of output images is shown in Figure 4.31 with curves represented using mutually exclusive colors, similar to Figure 4.7. Recall that an indication for a semantically meaningful vector structure is that visually continuous curves are stored using a single curve. This can be investigated for each method in Figure 4.31, by checking if visually continuous curves have a constant color (i.e., are represented using a single curve). Additionally, the curve structure should match the one in Figure 4.31a.

Figure 4.31 shows that most methods produce a vector structure that is somewhat similar to the ground truth, with varying quality and the methods by Egiazarian et al. [2020], Mo et al. [2021] not performing favourably. The exception is the method by Puhachov et al. [2021], which produces curves that are significantly smaller than the ground truth. This is due to the fact that its output primitive does not match the primitives used in the ground truth and is therefore split into a sequence of cubic bezier curves (see Section 4.3.1). In Figure 4.32, it can be seen that the vector structure of the output of this method before being split into cubic bezier curves contains curves that can be considered more semantically meaningful. However, they are significantly longer than the ground truth, confirming the mismatch between the primitives used by this method and the ground truth. Furthermore, the over-parameterization of primitives can be especially seen on curves that are supposed to be rather straight, e.g. the contours of the blade.

While Figure 4.31 seemingly shows that the vector structure produced by the methods match the ground truth, zooming into details of the image proves the contrary. All methods share the property that results appear to be visually correct at first glance, but 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.

The method by Mo et al. [2021] is similar to the method developed in this work in that

(a) Input raster image.

(b) Output of the developed method.

(c) Output of AutoTrace [Weber, 2002].

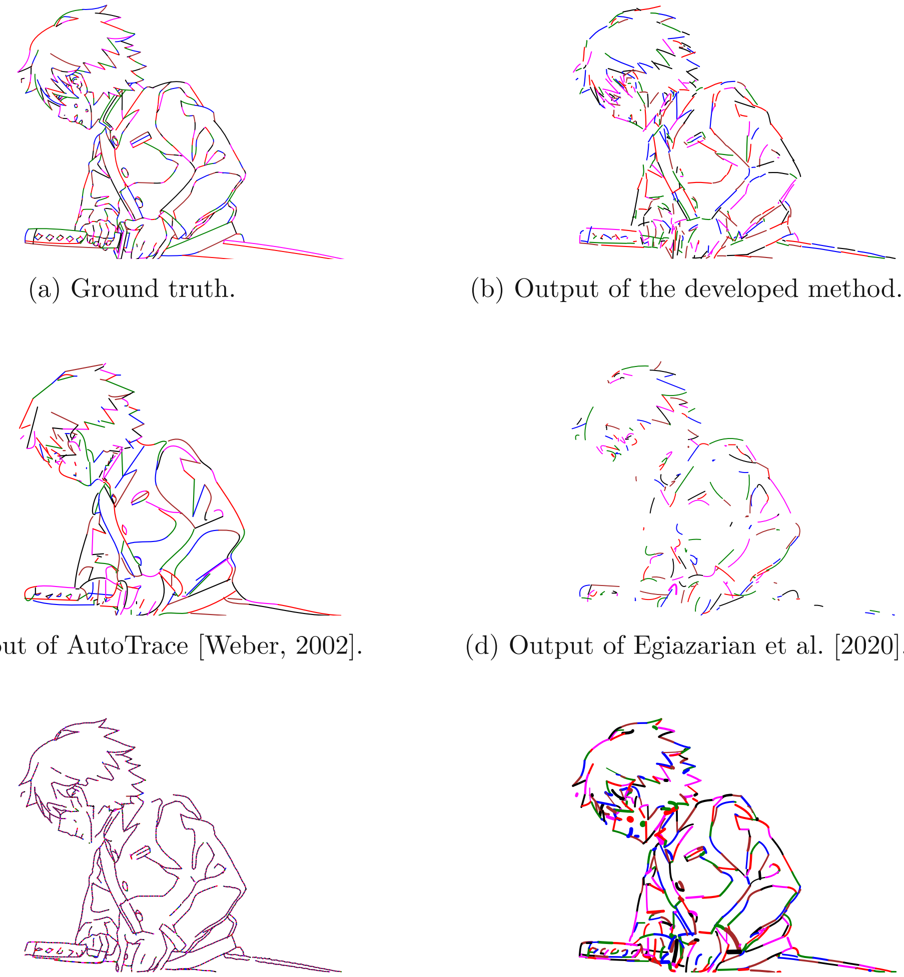
(d) Output of Egiazarian et al. [2020].

(e) Output of Puhachov et al. [2021].

(f) Output of Mo et al. [2021].

Figure 4.30: 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.

it faithfully reconstructs curves, but fails to preserve the constant stroke width. The methods by Weber [2002], Puhachov et al. [2021] 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 [Weber, 2002]. However, the produced vector structure is ultimately far from the ground truth in Figure 4.33a.



(a) Ground truth.

(b) Output of the developed method.

(c) Output of AutoTrace [Weber, 2002].

(d) Output of Egiazarian et al. [2020].

(e) Output of Puhachov et al. [2021].

(f) Output of Mo et al. [2021].

Figure 4.31: The vector structure behind the images in Figure 4.30 revealed by representing each curve with a mutually exclusive color.

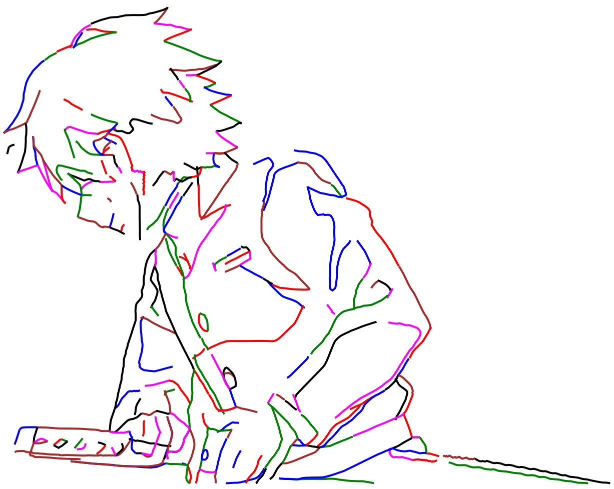
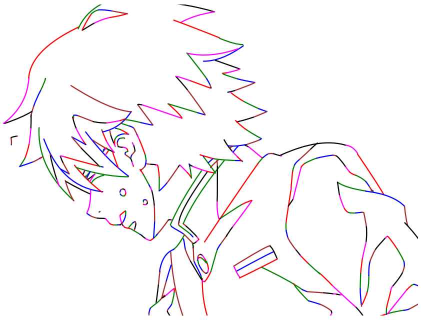


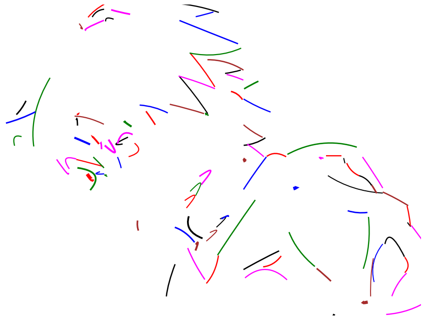
Figure 4.32: The output of [Puhachov et al., 2021] in Figure 4.31e without splitting the output primitive into cubic bezier curves.



(a) Ground truth.

(b) Output of the developed method.

(c) Output of AutoTrace [Weber, 2002].



(d) Output of Egiazarian et al. [2020].

(e) Output of Puhachov et al. [2021].

(f) Output of Mo et al. [2021].

Figure 4.33: The vector structure images in Figure 4.31 at high zoom level to reveal differences in the details.

(a) Output given a non-binarized standard-resolution input image.

(b) Output given a non-binarized high-resolution input image.

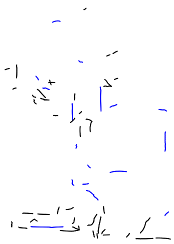
(c) Output given a binarized standard-resolution input image.

(d) Output given a binarized high-resolution input image.

Figure 4.34: The outputs of the method developed in this work given different versions of the input image shown in Figure 4.1a. Note that the outputs stay remarkably consistent given changing input images.

Robustness to the input image

Recall that the results of the methods by Puhachov et al. [2021], Mo et al. [2021], Weber [2002] were produced given a binarized version of the input image shown in Figure 4.1a. This is due to the fact that results on non-binarized input images were of considerably worse quality. In contrast, the method developed in this work is remarkably robust to the input image with respect to binarization and resolution, as Figure 4.34 shows. On the other hand, Figure 4.35 shows how brittle prior work is to non-binarized input images at the standard resolution considered in this work. While Figure 4.35 only shows examples of the method by Puhachov et al. [2021], the other methods similarly break down for non-binarized input images.



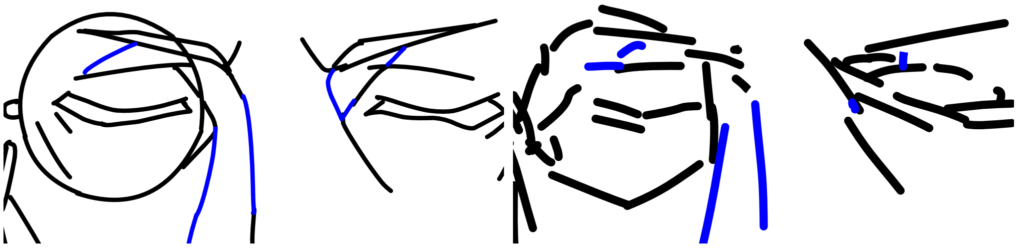
(a) Output given a non-binarized standard-resolution input image.

(b) Output given a non-binarized high-resolution input image.

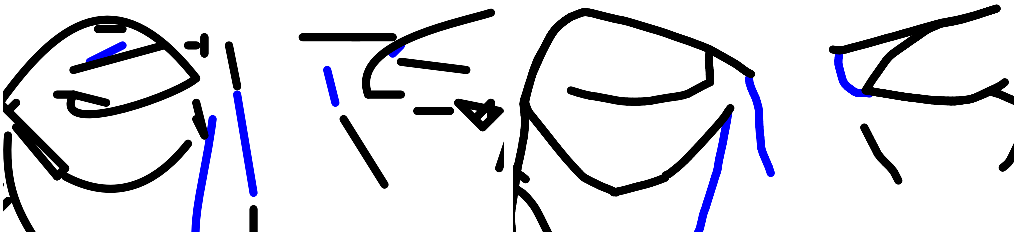
(c) Output given a binarized standard-resolution input image.

(d) Output given a binarized high-resolution input image.

Figure 4.35: The outputs of the method by Puhachov et al. [2021] given different versions of the input image shown in Figure 4.1a.



(a) Input image. (b) Output of the developed method.



(c) Output of AutoTrace [Weber, 2002]. (d) Output of Puhachov et al. [2021].

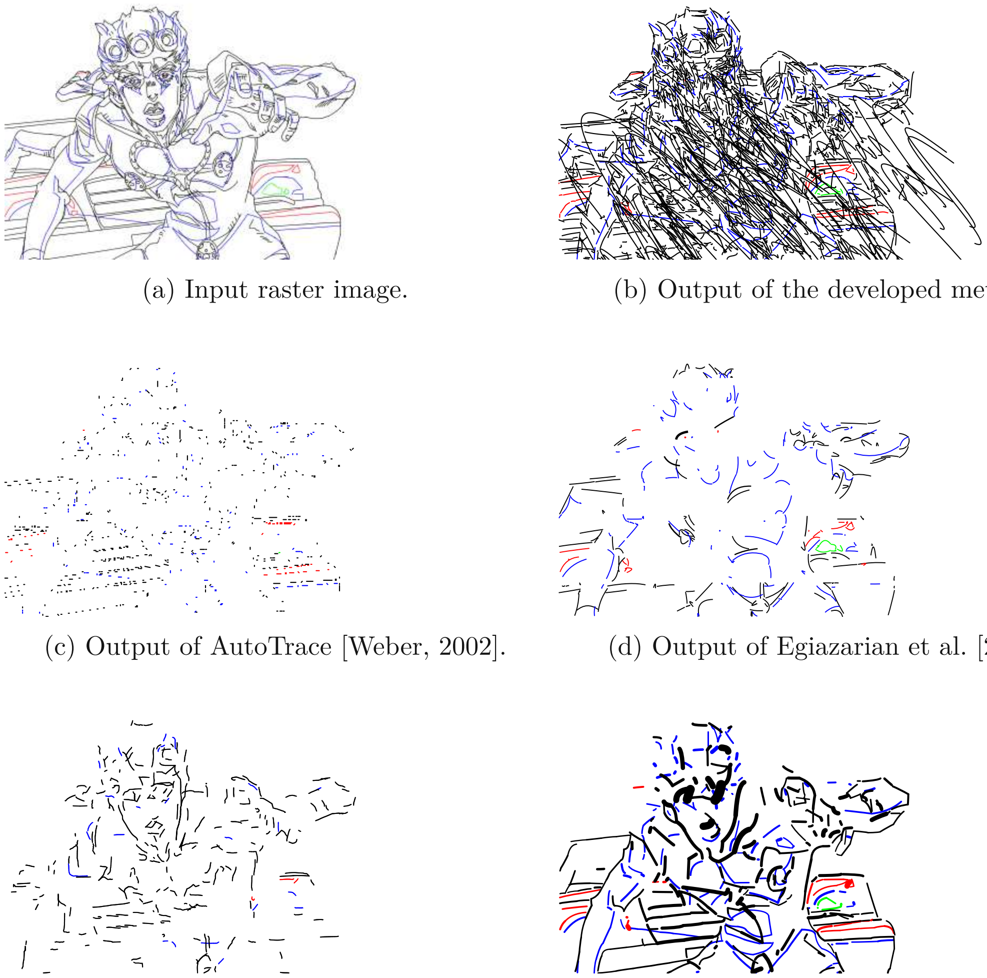
Figure 4.36: Outputs of the method developed in this work and the methods by Weber [2002], Puhachov et al. [2021] on an input image region containing high-curvature shapes such as circles. It can be seen that every method struggles to reproduce the circle with high curvature.

Low curvature

Both the method developed in this work as well as the methods by Weber [2002], Puhachov et al. [2021] produce the best results for shapes with a low curvature, which make up a large part of the images. The methods by Weber [2002], Puhachov et al. [2021] additionally produce their best results on longer curves. Figure 4.36 shows outputs for these methods on an input image with high curvature curves in the eye region. Note that again, the input was binarized for the methods by Weber [2002], Puhachov et al. [2021]. It can be seen in the eye region that the methods struggle to reconstruct the nearly perfect circle in the ground truth and instead approximate it by a sequence of low-curvature curves. As an aside, it is impossible to represent a perfect circle using a cubic bezier curve, but as the ground truth shows, it is possible to approximate a perfect circle in a way that is visually indistinguishable from it [Dokken et al., 1990].

Catastrophic failure

It is important to mention that there are cases where the method developed in this work catastrophically fails. This happens for one Tonari clean animation frame in the test dataset, which proves particularly challenging for all methods, as Figure 4.37 shows. It seems that the image contains too many small and detailed curves for methods to be able



(a) Input raster image.

(b) Output of the developed method.

(c) Output of AutoTrace [Weber, 2002].

(d) Output of Egiazarian et al. [2020].

(e) Output of Puhachov et al. [2021].

(f) Output of Mo et al. [2021].

Figure 4.37: The output images of the methods studied in this work given the challenging input example. Note that all methods fail reconstructing the input image, with the method by Mo et al. [2021] performing the best.

to reconstruct it properly at the standard resolution. It can be seen that the methods by Egiazarian et al. [2020], Weber [2002], Puhachov et al. [2021] simply ignore large parts of the detailed curves, arguably leading to a cleaner image than the output image of the method developed in this work. The method by Mo et al. [2021] seemingly performs the best for this challenging input image.

Two reasons can be identified which lead to this catastrophic failure. Firstly, the marked-curve reconstruction model was trained on a considerably small dataset and thus likely did not encounter an input image containing curves similar to the one in this challenging example. Secondly, a disadvantage of the formulation of the iterative curve reconstruction algorithm (see Algorithm 4.1) is that it keeps invoking the reconstructing curves until a specific number of black pixels are covered, even if the output curves are not meaningful anymore. The output could likely be improved by stopping the generation earlier or manually removing the superfluous curves. However, implementing such a quick fix automatically in the algorithm would harm generalization of the method.

For completeness, Figures A.5 to A.7 show further results of the methods studied in this work on both the Tonari and the SketchBench test dataset. Note that, again, the input images are binarized for the methods by Puhachov et al. [2021], Weber [2002], Mo et al. [2021].

4.4 Ablation

This section provides additional insights and context for the developed line-art image vectorization method. Section 4.4.2 describes model architectures explored early in the work and derives the chronology to the current marked-curve reconstruction model. Additionally, Section 4.4.3 provides an ablation study of different configurations of the marked-curve reconstruction model.

Note that, similar to the experiments in Section 4.3, all models are trained with the number 1234 used to seed random number generators.

4.4.1 Setup and Limitations

Due to computational efficiency and experiment design reasons, there are significant differences of the model comparisons in this section to Section 4.3.4. First, unless specified differently, the relevant metric used to compare models is the *single* curve IoU, i.e., the IoU between a single randomly chosen curve and its reconstruction by the model. Second, the metric is calculated on the *validation* dataset. Both of these decisions greatly reduce the runtime needed for metric computation, thereby enabling it to be run alongside training. Furthermore, since the model is applied iteratively on single curves each, the single curve IoU serves as a passable approximation of model performance. Additionally, this serves to prevent a leak of the test dataset into model training by it being used to guide the model design process.

Keep in mind that the models were trained on a rapidly changing codebase, severely limiting the generalizability of comparisons. In particular, metrics such as the single curve IoU are not comparable between changed model architectures.

Note that this section provides model comparisons as a post-hoc artifact of the model architecture design chronology. Thus, not all possible permutations of configurations are tried. Furthermore, of the permutations that were tried, only a small, selected amount of results is shown, as a lot more training runs were executed.

4.4.2 Earlier Architectures

This section gives the chronology of earlier architectures explored during the work on this thesis. This provides an indication of which approaches turned out to be successful and some indications on how the architecture could be improved.

Note that in general, the output images are displayed in raster format in this section.

Line-art raster image autoencoder

Initially, in order to evaluate the feasibility of reconstructing line-art images, a naive experiment of training a raster line-art image autoencoder (see Section 2.3.6) was conducted. Loosely motivated by the transferability of hyperparameters of simpler models to more complex models [Yang et al., 2021], this experiment also served as a starting point for the setting of hyperparameters in the model design phase.

The encoder consists of five stacked convolutional layers with a kernel size of 3, a stride of 2, padding of 1 and a filter size starting at 32 and doubling until reaching 512. The length of the latent vector is set to 8 to simulate the 8 control point parameters of a cubic bezier curve. The decoder is symmetrical to the encoder and consists of 5 transposed convolutional layers. All layers except the latent vector layer use the ReLU activation function, with the former using tanh (see Equation (2.2)). The autoencoder is trained on raster images of 64x64px resolution using the RGB color model consisting of up to 15 synthetically generated random black lines and white background with a batch size of 144 images. At each training iteration, a new batch of images is synthesized.

Of primary interest in this experiment was to determine a suitable learning rate and loss. With a learning rate of $\eta = 5 * 10^{-4}$ and the dice loss defined in Equation (2.16), the model quickly converges to reconstruct the lines with almost perfect accuracy after just 8 epochs. Figures 4.38d and 4.39c show the resulting images after training to convergence. Keep in mind that it did not see any image more than once, hence it likely did not overfit to the training data.

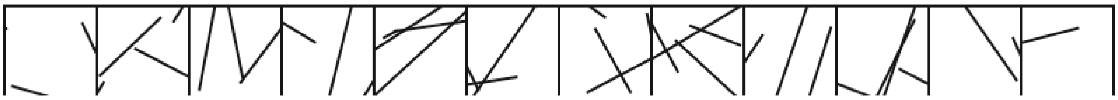
Other than the dice loss, models were also trained using the Huber loss defined in Equation (2.12) and the binary cross-entropy loss defined in Equation (2.14). With the Huber loss, the model converges the fastest, but lands in a local minimum of disregarding the color and reconstructing with significant *smear* and inconsistencies. Models trained with the binary cross-entropy loss perform almost as well as ones trained using the dice loss, but take more iterations to converge. Hence, the dice loss was chosen as starting point for successive experiments.

Other than losses and learning rates, interesting findings include that the model diverges without batch normalization [Ioffe and Szegedy, 2015]. Furthermore, setting the activation function of the latent layer to ReLU allowed the model to converge at a higher learning rate of $\eta = 5 * 10^{-3}$.

Furthermore, there was an attempt to train a VAE by adding a KL divergence [Kullback and Leibler, 1951] loss to the dice loss. The results in Figure 4.40 show that the model converged but to a worse quality than without the KL loss. Similar results were obtained for the binary cross-entropy and Huber loss. Since there is no need to sample outputs from the latent space, the VAE formulation can be safely ignored.

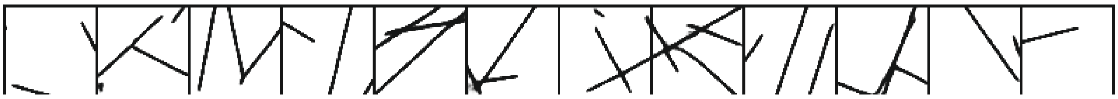
Im2Vec

Im2Vec [Reddy, 2021] was chosen as initial model architecture for experiments on line-art image vectorization because it was a recent method with publicly available code, model and dataset. Recall that Im2Vec is an encoder-decoder architecture, which is depicted in Figure 4.41. The raster input image is encoded to a latent vector of length 128 using a CNN. A RNN is used to produce a predetermined number of latent vectors for each

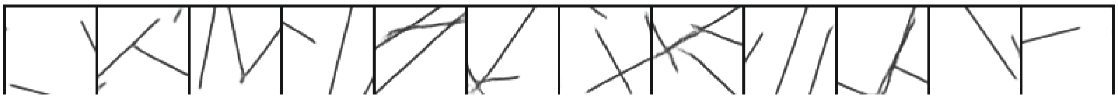


(a) The input images.

(b) The output images using Huber loss.

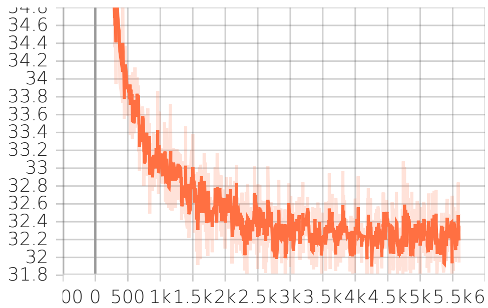


(c) The output images using binary cross-entropy loss.

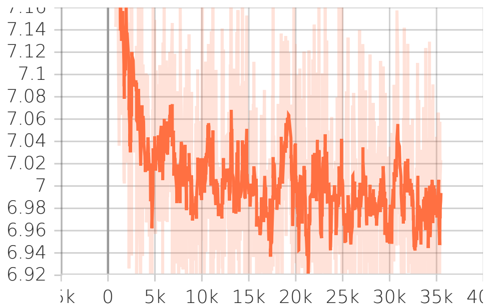


(d) The output images using Dice loss.

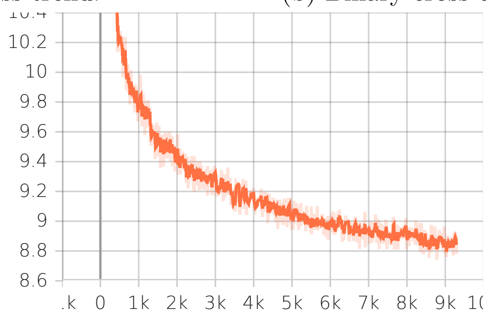
Figure 4.38: Output images of the autoencoder models trained on random line images with different loss functions.



(a) Huber loss trend.

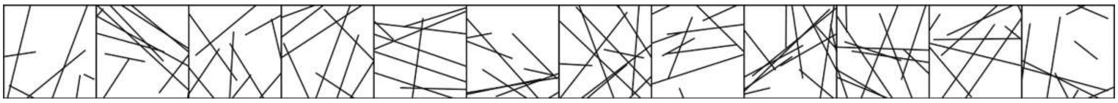


(b) Binary cross-entropy loss trend.



(c) Dice loss trend.

Figure 4.39: Trends for the losses used to train autoencoders.



(a) The input images.

(b) The output images.

Figure 4.40: A VAE trained on random line-art raster images.

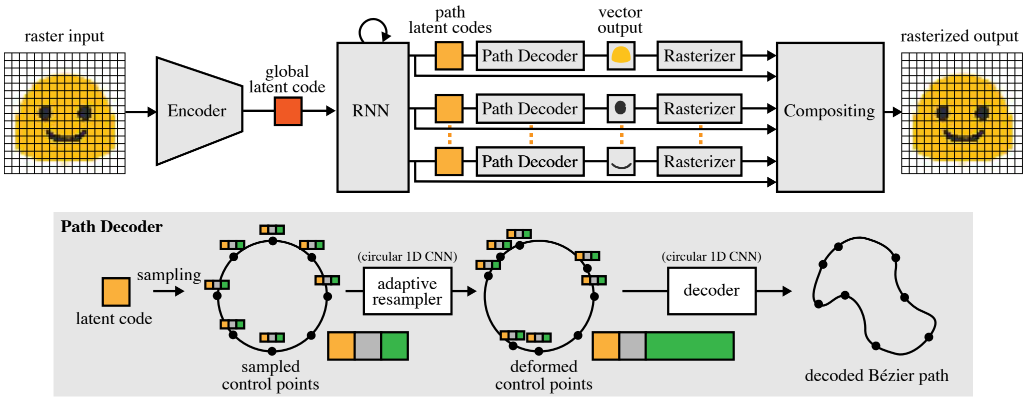
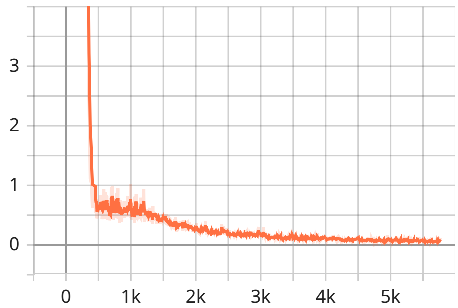


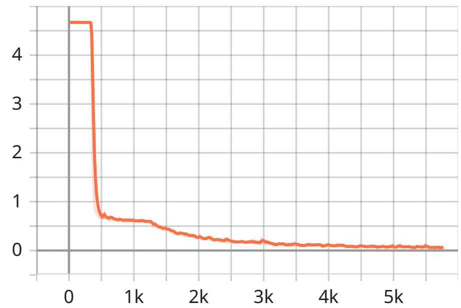
Figure 4.41: The architecture of Im2Vec [Reddy, 2021].

output shape. These path latent vectors are then decoded into vector primitives using 1-dimensional convolutional layers. Figures 4.42 and 4.43 show the provided results for the publicly available model on a test dataset consisting of emoji glyphs. Note that, in general, input images and output images are shown as a sequence of a batch. An attempt was made to reproduce these results using the publicly available code and dataset. However, the reproduction converged to a different local minimum of disregarding the green eyes, which can be seen in Figures 4.44 and 4.45.

Reproduction of Im2Vec [Reddy, 2021] was achieved by disabling the adaptive resampler. Note that Im2Vec uses closed sequences of bezier curves with a uniform fill color as primitives to reconstruct images. The control points for these bezier curve are sampled with a constant interval. The model contains an adaptive resampler as additional step, which trains the model to change the interval of these control points depending on the local structure of input shapes. In other words, it trains the model to use quadratic bezier curves or even lines for simple shapes, while using cubic bezier curves for complex shapes. Figures 4.46 and 4.47 show that the results with the adaptive resampler disabled are very close to the input images, with a smooth loss progression. Hence, the adaptive resampler was discarded from all further runs. The configuration for this reproduction is provided in the open source repository of this work.

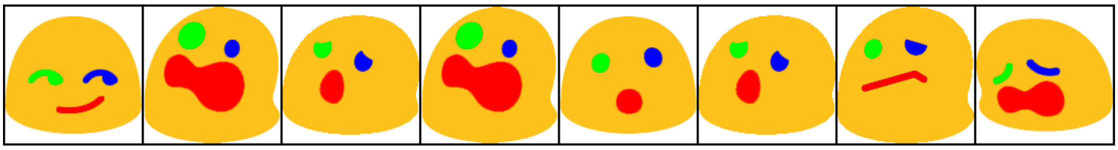


(a) The training loss.

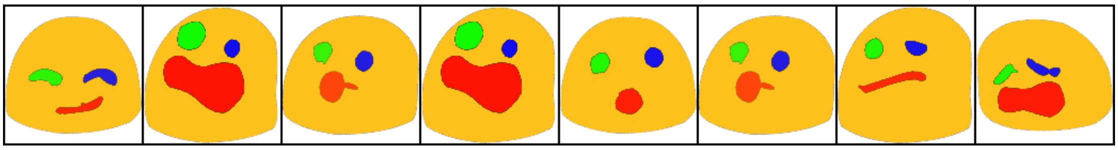


(b) The validation loss.

Figure 4.42: The training and validation loss of the model provided by Im2Vec [Reddy, 2021].

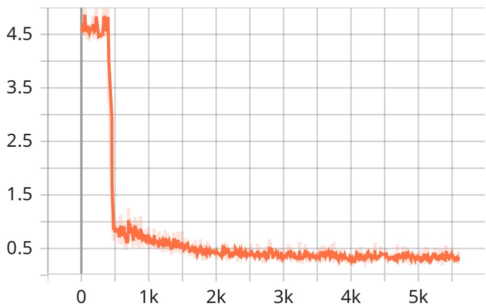


(a) The input images.

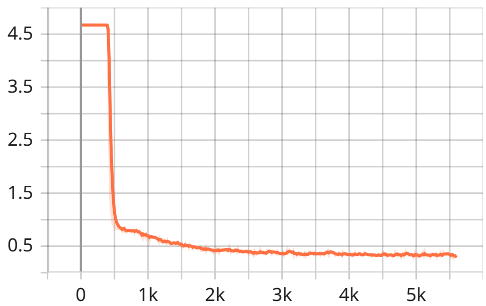


(b) The output images.

Figure 4.43: The test input images and the reconstructed output images of the model provided by Im2Vec [Reddy, 2021].

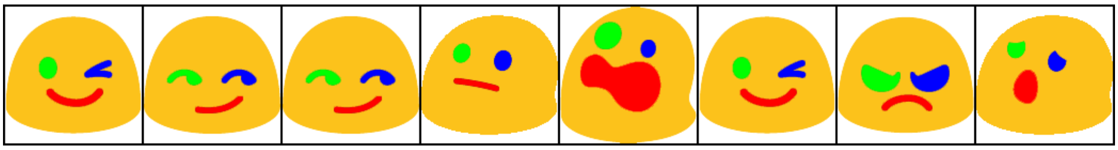


(a) The training loss.

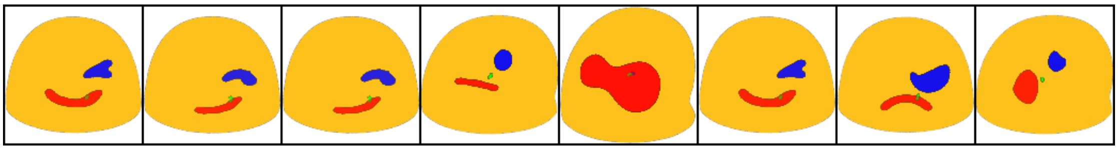


(b) The validation loss.

Figure 4.44: The training and validation loss of the attempted reproduction of [Reddy, 2021] using its provided code and dataset.

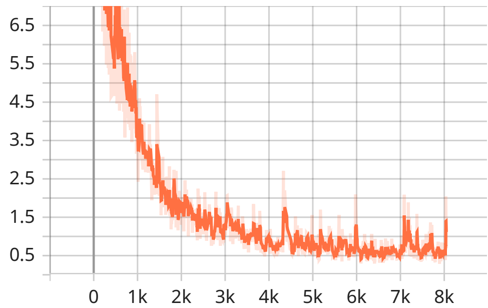


(a) The input images.

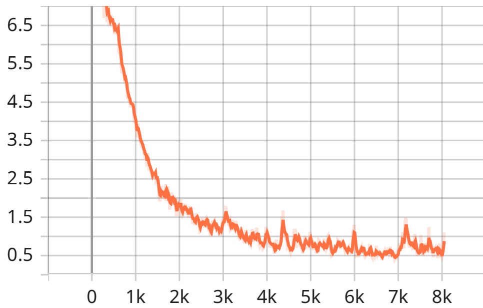


(b) The output images.

Figure 4.45: The test input images and the reconstructed output images of the attempted reproduction of [Reddy, 2021] using its provided code and dataset.

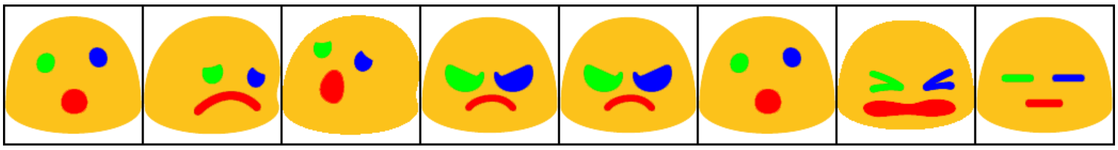


(a) The training loss.

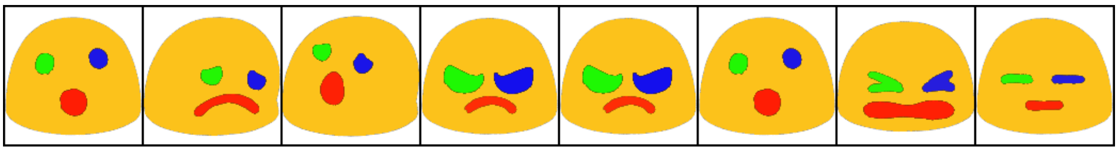


(b) The validation loss.

Figure 4.46: The training and validation loss of the attempted reproduction of [Reddy, 2021] using its provided code and dataset.

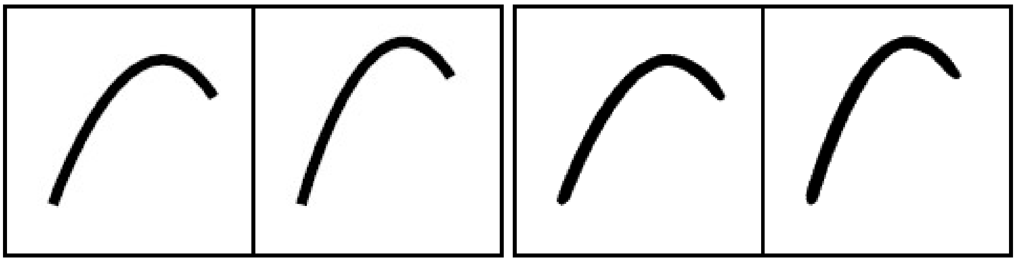


(a) The input images.



(b) The output images.

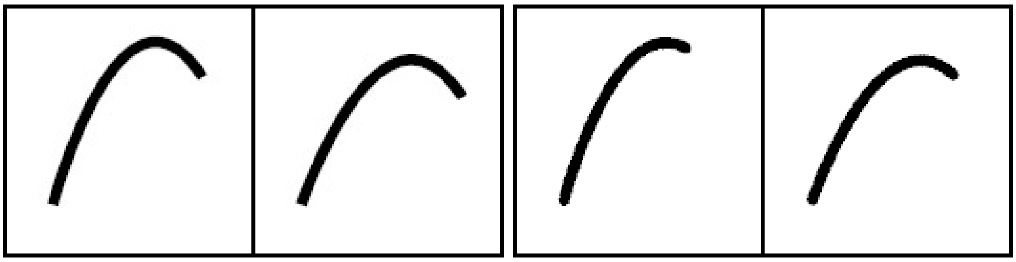
Figure 4.47: The test input images and the reconstructed output image of the attempted reproduction of [Reddy, 2021] using its provided code and dataset.



(a) The input images.

(b) The output images.

Figure 4.48: The input images and the reconstructed output images of [Reddy, 2021] trained on images with a single cubic bezier curve.



(a) The input images.

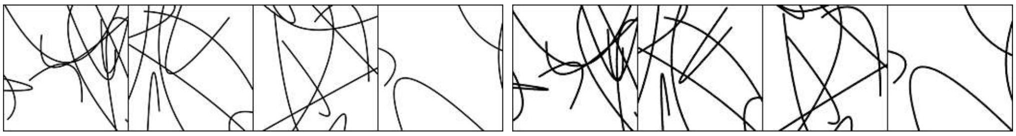
(b) The output images.

Figure 4.49: The input images and the reconstructed output images of [Reddy, 2021] restricted to use quadratic bezier curves and trained on images with a single cubic bezier curve.

With the Im2Vec [Reddy, 2021] model reproduced, the next step was to train it on clean line-art images. As Figure 3.1 shows, this could not be achieved as the model diverged. It is safe to assume that the model architecture is not suited for a large number of output primitives. Hence, the complexity was reduced and the model was trained to reconstruct a single quadratic bezier curve instead. As Figure 4.48 shows, it successfully converged to reproduce the curve. It is important to note that the training dataset consisted only of 2 images, intentionally letting the model overfit to it.

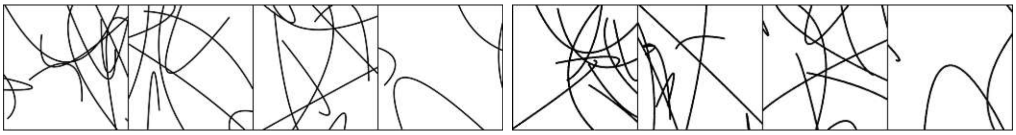
Notice that the reconstructed curves in Figure 4.48 do not exhibit constant stroke width. This is due to the fact that the Im2Vec model was not able to learn to reconstruct a single curve using a single bezier curve only and instead represents this curve using a filled sequence of multiple bezier curves. To prevent this over-parameterization, the output primitive was restricted to be a single quadratic bezier curve. Figure 4.49 shows that it was possible to train such a model, although the curve end points do not sufficiently match the input shapes.

Since the model architecture with output primitives restricted to quadratic bezier curves worked, the model was scaled up to reconstruct images containing multiple curves instead of a single curve. Again, the model is trained on a small dataset to intentionally let it



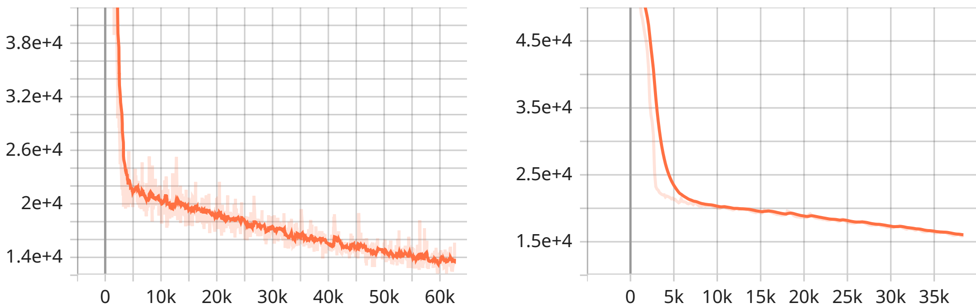
(a) The input images. (b) The output images.

Figure 4.50: The test input images and the reconstructed output images of [Reddy, 2021] restricted to use quadratic bezier curves and trained on images with multiple cubic bezier curves.



(a) The input images. (b) The output images.

Figure 4.51: The test input images and the reconstructed output images of [Reddy, 2021] restricted to use quadratic bezier curves and trained without overfitting on images with multiple random cubic bezier curve.



(a) The training loss. (b) The validation loss.

Figure 4.52: The train and validation loss trends for the model shown in Figure 4.51. The similarity of the validation loss to the train loss shows that the model did not overfit to the training data.

overfit to the data. Figure 4.50 shows that it is possible to scale up the model to multiple curves.

Since a model was successfully overfit to reproduce images with multiple quadratic bezier curves, the next step was to ascertain whether this is still possible without overfitting. For this purpose, a model was trained on 500 images, which produces passable results, as Figures 4.51 and 4.52 show.

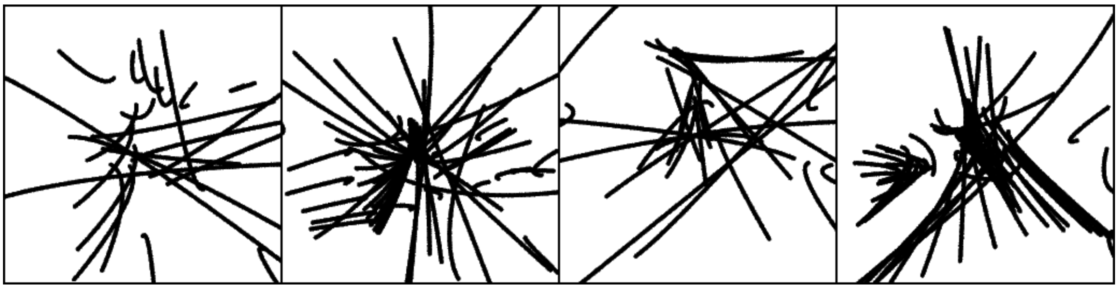


Figure 4.53: Output of a model trained to reconstruct input image containing 25 quadratic bezier curves.

Iterative single curve reconstruction model

Since the custom model architecture loosely based on Im2Vec [Reddy, 2021] developed in Section 4.4.2 was successfully used to train a model to reconstruct multiple randomly generated quadratic bezier curves, the next step is to scale this up to the roughly 1,000 curves required in a clean animation frame. However, training a model to output more than 15 curves proved unsuccessful. Figure 4.53 shows the results of a model trained on 25 curves. It is clear that the architecture is not suited for such a high number of output curves, since the output is generated in a single invocation using a RNN [Hochreiter, 1991, Bengio et al., 1994]. Hence, the architecture was changed to an iterative model, i.e., a model that only generates a single curve per invocation.

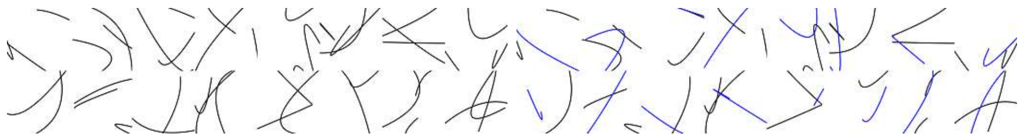
This is done by changing the output $\hat{\mathbf{y}}$ to consist only of the parameters of a single curve. The input to the model is a raster image $\mathbf{X}$ displaying multiple random curves and a raster image displaying already reconstructed curves, also called *canvas* image. Given this input, the model is trained to output curve parameters $\hat{\mathbf{y}}$ that match the ground truth curve $\mathbf{y}$. The ground truth curve is defined as the remaining curve that fits the generated curve the most. This iterative model can then be used to vectorize an entire image consisting of line art by iteratively invoking it to reconstruct a remaining curve until none are left.

The first step is to reproduce the result of the full model on input images of 15 curves using the iterative model. Once that works, it should be easy to scale up to more than 15 curves. This reproduction proved tricky. Figure 4.54 shows the results of an iterative model overfit to a small training dataset. Note that input and output images are displayed as a sequence of 4 images of 64x64px resolution, respectively. As in Figure 4.41, the input images of this section are raster images containing curves. In contrast, the output images now display the canvas image using black curves and the curve output by the model $\hat{\mathbf{y}}$ as a blue curve. Note that this represents only a single invocation of the model.

While the overfitted model produced a perfect reconstruction, training a model without overfitting on quadratic bezier curves was unsuccessful. Figure 4.55 shows the output of the best attempt after 3,000 iterations. Note that Figure 4.55 shows 16 input and output images of resolution 64x64px in 2 rows of 8 images each, respectively. It can be seen that

(a) Input images. (b) Output images (canvas images in black).

Figure 4.54: The output of the iterative model overfit to a small dataset of images consisting of quadratic bezier curves.



(a) Input images. (b) Output images (canvas images in black).

Figure 4.55: The output of the iterative model trained without overfitting on images consisting of quadratic bezier curves.

the only curve that is correctly reconstructed is one that resembles a straight line.

Since the iterative models trained without overfitting produced only straight lines correctly, the task was simplified to reconstruct lines instead of quadratic bezier curves. Figure 4.56 shows that a model could be somewhat successfully trained to achieve this. Additionally, Figure 4.57 shows the smooth progression of both the loss and the single curve IoU used to measure the model during training. Furthermore, the progression on the validation set looks similar to the training set, showing that the model did not overfit.

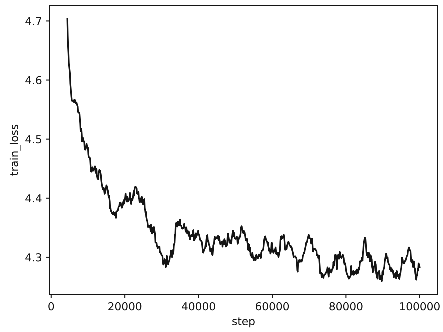
There are some changes to the model architecture which enabled the result in Figures 4.56 and 4.57. Most importantly, the loss was changed. It now consists of a combination of a raster-based and a vector-based loss, with the raster-based loss consisting of the dice and binary cross-entropy loss as motivated in Section 4.4.2 and the vector-based loss consisting of the MSE between the generated curve $\hat{\mathbf{y}}$ and the ground truth curve $\hat{\mathbf{y}}$. Additionally, as motivated by Section 4.4.2, the KL divergence loss was removed as it harmed results and served no purpose for this method. Moreover, the Im2Vec [Reddy, 2021] architecture calculates the loss at multiple resolutions of the input and the output image, apparently to derive a stronger gradient signal. However, this did not change results in the conducted experiments and was removed due to high computational requirements. Finally, in addition to the CNN encoder of the input and the canvas image, a Transformer [Vaswani et al., 2017] model was added to encode the vector paths of the canvas image.

One remaining issue of the models shown in Figure 4.56 is that the reconstructed primitives are often too short. This was improved by two further changes to the model architecture: Firstly the Transformer-based [Vaswani et al., 2017] path encoder was removed. Secondly, the ground truth target curve $\mathbf{y}$ was fixed to be the longest curve inside the image, which led to a smoother loss landscape. It seems that structuring the

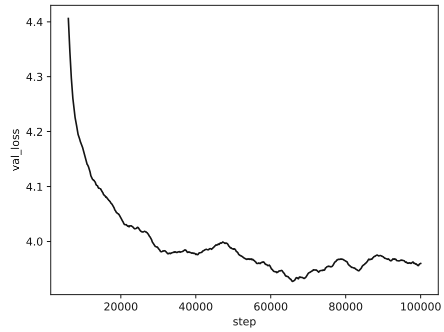
(a) Input images.

(b) Output images (canvas images in black).

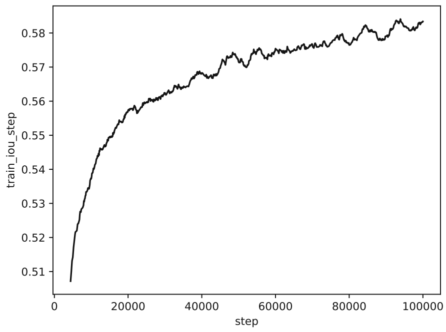
Figure 4.56: The output of the iterative model trained without overfitting on images consisting of lines.



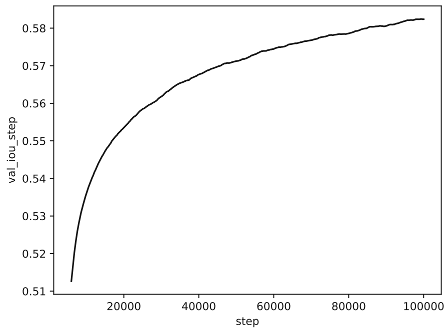
(a) Train loss.



(b) Validation loss.

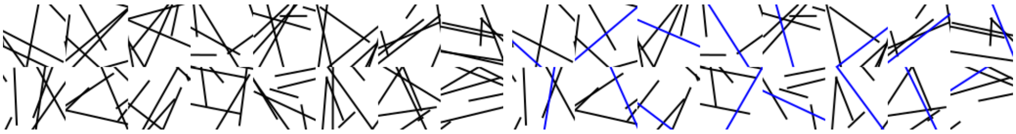


(c) Train single curve IoU.



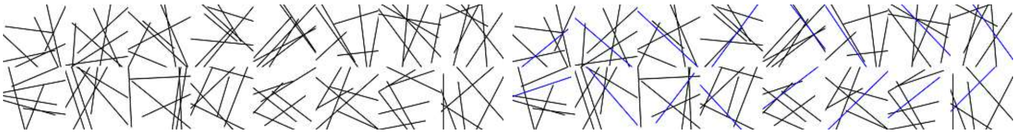
(d) Validation single curve IoU.

Figure 4.57: Loss and single curve IoU for the iterative model trained without overfitting on images consisting of lines.



(a) Input images. (b) Output images (canvas images in black).

Figure 4.58: The output of the iterative model trained on images consisting of lines.



(a) Input images. (b) Output images (canvas images in black).

Figure 4.59: The output of the iterative model trained on images consisting of lines with a resolution of 128x128px.

(a) Input images. (b) Output images (canvas images in black).

Figure 4.60: The output of the iterative model trained without overfitting on images consisting of quadratic bezier curves.

target this way is necessary, as the model does not have sufficient information on which curve to reconstruct otherwise.

Figure 4.58 shows that a model trained using the improved architecture generated primitives that fit the length of the ground truth. This architecture was further scaled up to a higher resolution of 128x128px, as shown in Figure 4.59. Note that the model architecture was still not stable enough to be properly scaled up to quadratic bezier curves, as Figure 4.60 shows.

After the model being somewhat successfully trained, it was evaluated on reconstructing a whole image. In other words, an algorithm starts with an empty canvas image and iteratively applies the single curve reconstruction model to reconstruct the longest remaining curve in the input image until none are left. Figure 4.61 shows this process for the case of lines. As can be seen, the image consisting of 4 random lines is reconstructed with some quality, albeit not perfectly. Figure 4.62 shows the same for a model trained on quadratic bezier curves. While the model outputs look promising, there are also frequent failure cases for both line and quadratic bezier curve inputs, one of which is shown in Figure 4.62. This may be due to multiple factors, with the main cause assumed to be that the information on which curve to reproduce is too ambiguous for the model, which harms the training process.

(a) Input. (b) Iteration 1. (c) Iteration 2. (d) Iteration 3. (e) Result.

Figure 4.61: The vectorization process of the single curve reconstruction model on an input image consisting of 4 random lines. The model is invoked sequentially until the number of output curves reaches 4. At each time step, the previously reconstructed curves serve as canvas image.

(a) Input image 1. (b) Final result 1. (c) Input image 2. (d) Final result 2.

Figure 4.62: Two example results of the iterative model applied to images consisting of quadratic bezier curves. The first reconstruction succeeds, while the second one fails.

Marked iterative model

As Figures 4.59 and 4.60 show, was possible to train an iterative model for lines using the designed architecture, but the results were not good enough to scale it up to quadratic - let alone cubic - bezier curves. Hence, the architecture was changed in order to reduce the complexity of the task for the model. The reduced task of the model is to reconstruct a single curve, with explicit information about the curve given to the model. This is in contrast to the model in Section 4.4.2, where the information about which curve to reconstruct is given implicitly by the curve length or not given at all. The explicit information is passed to the model via coloring the curve red, which can be seen in the input images of this section.

Figure 4.63 shows that this new task formulation enabled a model to be trained that reconstructs the target line nearly perfectly, reaching a high single curve IoU of 0.88. This could be successfully scaled up to quadratic bezier curves, as Figure 4.64 shows. This model also reached a high single curve IoU of 0.84.

While the models shown in Figures 4.63 and 4.64 give a good indication of the maximum potential of a line-art image vectorization model, they cannot be used for real-world data because they require the target curve to be fully marked to be able to reconstruct it. A more realistic formulation is that the target curve information is indicated by placing a mark at a single location of the curve, which reduces the heavy requirements for the

(a) Input images. (b) Output images.

Figure 4.63: The output of the marked iterative model trained on images consisting of lines.

(a) Input images. (b) Output images.

Figure 4.64: The output of the marked iterative model trained on images consisting of quadratic bezier curves.

(a) Input images. (b) Output images.

Figure 4.65: The output of the marked iterative model trained on images consisting of lines, with a single mark placed on the target curve instead of the entire curve being colored.

curve identification. Furthermore, the input image is now centered on the mark before being input into the CNN encoder. Figure 4.65 shows that a model for lines could be successfully trained using this formulation, reaching a single curve IoU of 0.79. Note that the mark is indicated as a red pixel on the target line. Accordingly, Figure 4.66 shows this model architecture scaled up to quadratic bezier curves, again reaching a single curve IoU of 0.79. Finally Figure 4.67 shows that the model architecture can be scaled to cubic bezier curves, which is the target primitive of this work. It reached a somewhat lower but passable single curve IoU of 0.72.

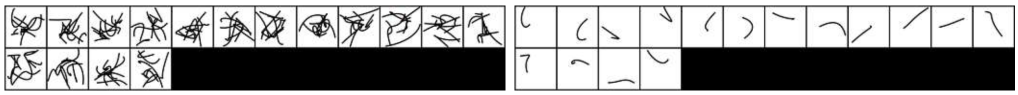
The model architecture derived in this section can be scaled up to cubic bezier curves as primitives and to a large number of curves due to the iterative nature. Furthermore, to give an example of how the model works, Figure 4.68 shows the process of vectorizing an entire input image consisting of 7 quadratic bezier curves, similar to Figure 4.61.

4.4.3 Marked-Curve Reconstruction Model configurations

The marked-curve reconstruction model architecture derived in Section 4.4.2 and shown in Figure 4.66 forms a good base for a line-art image vectorization algorithm. This section

(a) Input images. (b) Output images.

Figure 4.66: The output of the marked iterative model trained on images consisting of quadratic bezier curves, with a single mark placed on the target curve instead of the entire curve being colored.



(a) Input images. (b) Output images.

Figure 4.67: The output of the marked iterative model trained on images consisting of cubic bezier curves, with a single mark placed on the target curve instead of the entire curve being colored.

(a) Input. (b) $t = 1$. (c) $t = 2$. (d) $t = 3$. (e) $t = 4$. (f) $t = 5$. (g) $t = 6$. (h) End.

Figure 4.68: The vectorization process of the marked single curve reconstruction model on an input image consisting of 7 quadratic bezier curves. The model is invoked sequentially until the number of output curves reaches 7. At each time step, the previously reconstructed curves are removed from the canvas image and a mark is placed on a remaining curve.

provides an ablation of various different configurations of this model.

Setup

During the model design phase, the model was trained using different configurations. The single curve IoU is used as metric to compare these configurations. In order to achieve consistent comparability, this metric is collected at iteration 100,000 for every model. Note that this is not the final performance of the model, as results still improve after 100,000 iterations (see Figure 4.69). However, the rate of improvement decreases over time, leading to the performance at step 100,000 to be a passable approximation of final performance. This reduces the time needed to train the models.

Since the metric value at a single point $t = 100000$ exhibits a large variance, the average of the values of the last 30 steps leading up to $t = 100000$ is used as performance indicator. Again, like in Section 4.3.4, the IQR is provided as a scale measure. Keep in mind that contrary to Section 4.3.4, the distribution of the single curve IoU is over 30 iterations

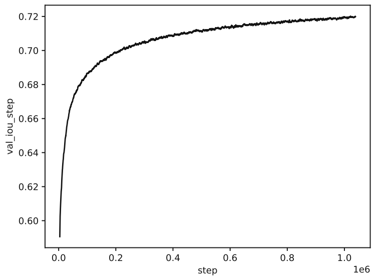


Figure 4.69: The validation single curve IoU for the marked reconstruction model trained on random cubic bezier curves. Note that while performance improves until the final step (over 1,000,000), the model already achieves 97% of its final performance at 200k steps and 94% of its performance at 100k steps.

and thus non-stationary. However, as the change in performance of the model does not change significantly within 30 iterations, this is not a problem in practice.

As in Section 4.3.4, the arrow in the metric column indicates whether higher or lower numbers are better, with the results sorted ascending. All values that are not statistically significantly different from the best value are highlighted. As in Section 4.3.4, the Wilcoxon-Mann-Whitney U-Test [Wilcoxon, 1945, Mann and Whitney, 1947] is used to ascertain statistical significance. For the interpretation of this test, again, note that it is now calculated between the distributions of the metric over the last 30 steps.

Training dataset

Note that in contrast to the models in Section 4.4.2, the input image resolution is increased to 512x512px in order to match real-world data more closely (see Section 4.2.4).

Synthetic data The models derived in Section 4.4.2 were trained and evaluated solely using synthetic data (see Section 4.2.2). Attempts to scale these models to the higher resolution of 512px failed, as all training runs diverged. Together with the fact that the models are intended to be used on human-generated images, this leads to the assumption that human-generated data needs to be added to the training dataset. The remaining question is what the ratio of synthetic data to human-generated data should be. Table 4.18 shows that a ratio of 1 to 5 performs better than an even match between synthetic and human-generated data. In conclusion, it seems that too much synthetic data is harmful for performance on human-generated images.

TU Berlin subset While the Tonari and SketchBench subsets are assumed to be essential for model performance on professional line-art images, it is questionable if the amateur sketches of the TU Berlin subset are necessary. Table 4.19 shows that the model actually diverges if the TU Berlin subset is not included, showing that it is necessary to

synthetic data ratio	Single curve IoU ↑ median	IQR
1:1	0.550	0.015
1:5	**0.608**	0.023

Table 4.18: Comparison of models trained with different amounts of synthetic data.

TU Berlin subset	Single curve IoU ↑ median	IQR
without TU Berlin	0.501	0.000
with TU Berlin	**0.613**	0.002

Table 4.19: Comparison of models trained with and without the TU Berlin subset.

data augmentation	Single curve IoU ↑ median	IQR
no data augmentation	0.610	0.002
data augmentation	**0.617**	0.002

Table 4.20: Comparison of models trained with and without data augmentation.

train a model successfully. Furthermore, this shows that the Tonari and SketchBench datasets are not sufficient to train a model. This is likely due to their small size. Another explanation would be that the TU Berlin subset serves as a kind of curriculum learning [Bengio et al., 2009], providing learning signals for simpler sketches to enable the model to reach a sufficiently optimized state to take advantage of the learning signals for the professional line-art images in the Tonari and SketchBench datasets.

Data augmentation As explained in Section 4.2.4, the training dataset is synthetically increased by applying data augmentation transformations on the human-generated subsets. Table 4.20 shows that this significantly improves the performance of the model.

Learning rate

A common question when training machine learning models is which learning rate to use (see Section 2.3.1). The learning rate affects the magnitude of parameter updates in a single iterations. Table 4.21 shows that the learning rate of $\eta = 5 * 10^{-4}$ derived

learning rate	Single curve IoU ↑ median	IQR
0.005	0.527	0.000
0.0005	**0.609**	0.008
0.00005	**0.610**	0.001

Table 4.21: Comparison of models trained with different learning rates

global pooling	Single curve IoU ↑ median	IQR
no global pooling	0.610	0.002
global pooling	**0.611**	0.001

Table 4.22: Comparison of a model trained with global pooling and a model trained by flattening the CNN output using an MLP layer.

in Section 4.4.2 performs the best, together with a lower learning rate $\eta = 5 * 10^{-5}$. Increasing the learning rate to $\eta = 5 * 10^{-3}$ leads to divergence. Hence, $\eta = 5 * 10^{-4}$ is maintained as learning rate for the model.

Global pooling

As described in Section 2.3.3, there are two ways of differentiably reshaping CNN outputs to a flat vector of a predefined size. One is to flatten the CNN output followed by an MLP layer. The other is to use global pooling, which does not require a fixed width and height to be set at training or inference time. The other quality of global pooling is that it contains fewer learnable parameters, which may be a disadvantage or an advantage depending on the model. Table 4.22 shows that the reduced parameter size is not a disadvantage and that the model with global pooling performs significantly better than the model without. Hence, since global pooling is more flexible with respect to the input image size and has less runtime and memory complexity than the MLP approach, it is used to reshape the CNN encoder output.

Loss

As the loss function acts as differentiable proxy of the task objective (see Section 2.3.2), it is an important decision in the model design. Since the model directly optimizes the loss function, different formulations of it might lead to better or worse performance at the intended task.

	Single curve IoU ↑	
	median	IQR
curve length loss weight		
curve length loss weight	0.602	0.002
no curve length loss weight	**0.610**	0.002

Table 4.23: Comparison of a model trained with the loss weighted by the curve length and a model trained without.

	Single curve IoU ↑	
	median	IQR
loss		
vector loss	0.560	0.010
vector + raster loss	**0.608**	0.023

Table 4.24: Comparison of a model trained with vector supervision only and a model trained with vector and raster supervision.

Weighing loss by curve length One shortcoming of the trained model is that curves are often slightly too short, leaving holes within curve sequences. One potential fix to this is to assign the curve length as weight to the loss, thus increasing the focus of the model on longer curves and decreasing it for shorter curves. However, as Table 4.23 shows, this decreases the performance of the model. As it does not improve the curve length, this loss formulation is not considered further.

Vector vs. raster supervision The combination of vector and raster-based losses is an important distinction of this method from prior work. Hence, it is important to ascertain whether it is actually essential for model performance. Table 4.24 shows that a model trained using the vector-based loss achieves a significantly lower single curve IoU than a model trained using a combination of a vector- and raster-based loss. Note that the metric used to measure performance itself is calculated on raster representations of vector structures, which intuitively shows why a raster-based loss is an important addition.

Binarization

As shown in Section 4.3.4, binarization of input images improves the results for some line-art image vectorization methods studied in this work. On the other hand, the method developed in this work performs remarkably similar when given binarized and non-binarized input images. To investigate whether this is just an artifact of the training data, two models are trained with binarized and non-binarized input images, respectively.

binarization	Single curve IoU ↑ median	IQR
binarized	**0.606**	0.004
non-binarized	**0.608**	0.023

Table 4.25: Comparison of models trained with and without binarization of input images.

Table 4.25 shows that there is no significant difference between those two models, leading to the conclusion that model performance does not depend on the binarization of input images.

CHAPTER 5

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. For this purpose, Chapter 2 provided the theoretical foundations and Chapter 3 explored existing line-art image vectorization methods. In order to answer the RQ1, Chapter 4 proposed a clean animation frame line-art image vectorization method and compared it with prior work on a test dataset provided by Tonari Animation. It could be shown that while the proposed method outperforms prior work at the default input image resolution, ultimately all line-art image vectorization methods work only in limited extent for clean animation frames, especially failing to properly reconstruct details and primitives with high curvature.

This chapter provides an analysis of advantages and disadvantages of the contributions in Section 5.1. Section 5.2 summarizes the limitations of this work. Finally, Section 5.3 lists potential future work relating to the research presented in this work.

5.1 Advantages

The developed line-art image vectorization method exhibits a number of advantages compared to existing methods. These encompass both general advantages and advantages relating to the application domain of clean animation frames.

Resolution-independence As shown in Sections 4.3.4 and 4.3.5, the performance of the developed method remains remarkably stable at different input image resolutions. In contrast, other state-of-the-art methods perform noticeably worse at lower resolutions. This makes the developed method uniquely suited to be applied to clean animation frames, which are drawn with a target resolution of 720x405px in the dataset considered in this work (see Section 4.2). Note that, as mentioned in Section 4.3.4, one potential way to apply other methods to input images at lower resolutions is to use super-resolution models [Dong et al., 2016] on the input images. However, these models would need to be successfully finetuned to high-resolution line-art images beforehand, which is an open research question on its own.

Keep in mind that the resolution-independence property was only tested at resolutions between 512px and 1024px. Resolutions smaller than 512px are unreasonably small to perform line-art image vectorization on.

Binarization-independence Similar to the input image resolution, the performance of the developed method remains remarkably stable on binarized and non-binarized versions of the input image. In contrast, methods such as the ones by Puhachov et al. [2021], Weber [2002] perform noticeably worse on non-binarized input images. This decrease in performance is especially pronounced for low-resolution non-binarized images (see Sections 4.3.4 and 4.3.5). For these kinds of input images, the method developed in this work is uniquely suited. For other cases, prior work might yield better results.

Nearly end-to-end differentiable The method developed in this work is designed to be as end-to-end differentiable as possible. This is done in order to improve the ease of adapting it to input images of other domains via finetuning (see Section 1.3). The only non-differentiable part is the iterative placing of marker pixels on curves to reconstruct (i.e., curve identification, as explained in Section 4.1.2). To make the model completely end-to-end differentiable, this part would need to be replaced with a learned model.

Easy manual fixing It is inevitable that output vector images will contain errors. These can be roughly divided into missing curves and incorrect curves. As described in Section 2.2.1, identifying missing curves is easier than fixing incorrectly reconstructed curves. Hence, the method is decomposed into a curve identification and curve reconstruction part. This way, missing curves (i.e., errors in the curve identification) can easily be fixed by placing a marker pixel on it and invoking the curve reconstruction part of the method (see Section 4.1.2).

Performance The developed method is the fastest deep learning-based method tested, being significantly faster than other methods (see Section 4.3.4). However, note that it is significantly outperformed by the traditional AutoTrace [Weber, 2002] algorithm.

On non-binarized low-resolution images, the method is 1.5x faster than the second-fastest deep learning-based method. However, note that most other methods do not perform well on such input images. These work better on binarized or high-resolution input images, where the runtime of the developed method stays remarkably stable, while the runtime of other methods drastically increases. This leads to a runtime that is up to 4.5 times faster than the second-best performing method.

Furthermore, the trained model is also the smallest of all deep learning models considered in this work, consisting of only 2.2 million parameters with a maximum GPU memory usage of 1008 MiB (see Table 4.11).

Furthermore, for accessibility and reproducibility, the code, model and parts of the dataset are publicly available at `https://github.com/nopperl/marked-lineart-vectorization`. Note that proprietary clean animation frames form a significant part of the training dataset, but the evaluation in Section 4.3 was also performed on a publicly available dataset.

Suited for large number of curves As described in Section 2.2.1, clean animation frames consist of a large number of curves. In order to handle this, the method is designed as an iterative algorithm. It shares this property with all other prior work evaluated in Section 4.3.

5.2 Limitations and Disadvantages

To better contextualize the contributions of this work, this section lists disadvantages and limitations.

5.2.1 Disadvantages

The developed line-art image vectorization method possesses a range of disadvantages which limit its application to real-world images.

Holes in curve sequences The developed method on average reconstructs curves smaller than they appear visually. This leads to a high number of small holes in curve sequences, which is very undesirable for clean animation frames. As Sections 2.2 and 2.2.1 describe, one step in the limited animation production workflow after the clean animation frame is drawn is the filling of colors in the regions defined by the clean animation frame. This can only be done efficiently if the regions are properly enclosed.

Limited semantic correctness As explained in Section 1.2, an important property of resulting clean animation frames is that their vector structure is semantically correct. This was attempted by training the marked curve reconstruction model using a vector-based loss. However, as Sections 4.3.4 and 4.3.5 show, similar to other evaluated methods, the outputs of the developed method do not sufficiently match the required semantic structure. This is especially the case for regions with small details such as eyes. These errors are especially undesirable, as clean animation frames need to be semantically correct for successive workflow steps, thus requiring laborious manually fixing.

Bias towards low curvature As Section 4.3.5 shows, the developed method possesses a bias towards generating primitives with low curvature, even if the target shape in the input image has a high curvature. It shares this property with other state-of-the-art methods. This harms the applicability of the method to images containing a high amount of high-curvature shapes.

Catastrophic failure As Section 4.3.5 shows, the developed method catastrophically fails to vectorize a challenging clean animation frame in the test dataset. This non-linear and seemingly random failure mode limits the applicability of this method. A potential cause for this failure is the low amount of training data, as such failures are common for underfit deep learning models.

5.2.2 Limitations

There exist limitations in the method design, data and evaluation setup as noted in Sections 4.1.2, 4.2.3, 4.3.2 and 4.4.1.

Section 4.1.2 describes limitations in the method design, which mainly consist of the method being unable to handle overlapping curves properly. In these cases, there exist pixels in the input raster image which belong to multiple curves. Once one of these curves is reconstructed, the overlapping pixels are removed, leaving behind a hole inside the other curves to which the pixel belongs. If this hole is large enough, it effectively splits the curve for the model, which will then reconstruct only the part until the hole. However, the amount of overlapping curves is rather small in the training and evaluation data, as shown in Table 4.3.

There is a considerable amount of limitations associated with the data used in this work, which are described in Section 4.2.3. The main limitation is the scarce amount of data used to train the model, which serves as a significant bottleneck for model performance. Furthermore the artist distribution for the Tonari clean animation frames is skewed, which harms generalization. Moreover, there are a range of irregularities in the Tonari clean animation frame subset, such as visually erased curves still being present in the vector structure, overlapping curves and curves with colors and stroke widths that do not conform to the clean animation frame schema defined in Figure 2.10.

A range of factors limit the evaluation, as described in Section 4.3.2. Most importantly, the evaluation is based on a subset of proprietary clean animation frames by Tonari Animation. However, this issue is solved by additionally computing all evaluation steps with the publicly available SketchBench dataset [Yan et al., 2020]. This includes a separate model trained without information leak from SketchBench test data. Additionally, there are limitations in the setup of prior works, such as the refinement algorithm by Egiazarian et al. [2020] lending an unfair advantage to their method, the method by Puhachov et al. [2021] requiring the proprietary Gurobi library [Gurobi Optimization, LLC, 2023] and AutoTrace [Weber, 2002] being run without GPU acceleration.

Furthermore, keep in mind that the line-art image vectorization method as presented in this work is limited to clean line-art images. However, due to the end-to-end differentiable nature of the method, Section 4.1.2 proposes ways to adapt it to inputs of different domains.

5.3 Future Work

There are numerous opportunities for future work relating to the research presented. These mainly pertain to the RO1 and include improvements to the developed method by changing the data used to train the model (see Section 5.3.1) or the design of the method architecture (see Section 5.3.2). Furthermore, an important potential extension is the adaptation of the developed method to new tasks (see Section 5.3.3).

Moreover, a straightforward option for future work regarding the RO2 (i.e., the evaluation) would be to assess how the developed line-art image vectorization method performs on different but adjacent domains (e.g., product sketches).

5.3.1 Data Improvements

As noted in Section 4.2.3, the scarce amount of high-quality data is the main limiting factor of this work. There are multiple potential ways to improve this issue. One is to train a general line-art image vectorization model on a large dataset consisting of multiple domains of line-art images such as technical line drawings, product sketches, amateur sketches, cartoons or illustrations. A successful clean animation frame vectorization model could then be achieved by finetuning the general model on a small clean animation frame dataset. For this approach to work, the training dataset of the general model needs to be several orders of magnitudes larger than the one considered in this work.

Another way to increase the available data is to consider more complex data synthesizing techniques. The synthetic data used in this work is quite trivial and devoid of any semantic structure. There may be heuristic optimization or deep learning-based techniques to synthesize data that matches clean line-art images more closely.

On the other hand, it may be possible to exploit the small available data in a better way. This might be achieved by a more complex mixture of the data, such as starting the training process with a high portion of TU Berlin amateur sketches and linearly reducing the contribution of this subset in the spirit of curriculum learning [Bengio et al., 2009]. Another possibility is to extract more structural features from the input raster image using pre-trained models, such as keypoint detection [Puhachov et al., 2021], object detection [Jocher, 2020], edge detection [Soria et al., 2020] or segmentation [Hu, 2020] models.

Furthermore, as noted in Section 4.2.1, an important decision is whether to consider curves or paths of curves as the graphical primitive for the model, with this work deciding for the former. An interesting question is how the methods studied in this work perform when using paths as primitive.

A dialectical approach would be to use both paths and curves as primitives and structure the model output hierarchically. In other words, the model could be trained to output either a single curve or a sequence (and possibly a hierarchy) of curves. This would be useful for domains in which hierarchical information is important, which is not the case for clean animation frames.

5.3.2 Architectural Improvements

There are multiple potential avenues for improving the architecture of the developed method. For one, in the iterative curve reconstruction algorithm, the input image is centered on the target mark prior to being input into the marked curve reconstruction model. It might also be useful to zoom the input image in the center. This leads to two benefits. Firstly, the model no longer receives distracting information about the periphery of the image when the task is to reconstruct only the curve at the center. Secondly, if the input image is of high resolution, the model would receive the zoomed in region at a higher resolution than in the standard algorithm design, potentially enabling it to better discern features about the curve to reconstruct. A potential disadvantage of this are long curves, which could get clamped by the zoom level. However, such curves are scarce in the data considered in this work.

Another possibility of improving the iterative curve reconstruction algorithm is to choose a better stopping criterion described in Section 4.1.2. It is currently defined as $T = \lfloor B*0.1 \rceil$, where B is the number of black pixels in the original image. This might be too high for some images and too low for others (such as the one shown in Figure 4.37). It might be possible to derive a stopping criterion that is better suited to the input image by directly predicting the optimal number of output curves using a small learned model instead.

Furthermore, it might be possible to extend the iterative curve reconstruction algorithm with more extensive post-processing. One possibility is to use the refinement algorithm of Egiazarian et al. [2020], which has the potential of significantly improving results (see Figure 3.2). However, it is not differentiable and has strong assumptions on the input image.

There are also potential improvements on the model side. As Sections 4.3.4 and 4.3.5 show, the vector structure of the model output is not sufficiently semantically correct. A safe assumption is that this is due to the vector-based loss not properly evaluating the output of the model. The vector-based loss used in this work is an even combination of the MSE and MAE between the output and ground truth curve parameters adapted from Egiazarian et al. [2020]. Other formulations might work better, such as weighting the MSE and MAE according to a linear schedule, aggregating the errors with a sum instead of the average or even differentiably rendering a heatmap of the curve parameters and using a raster image loss following the keypoint detection loss in Puhachov et al. [2021].

An important decision in the developed method is to use an iterative model. This is influenced by the inability of RNNs to output large sequences properly [Hochreiter, 1991, Bengio et al., 1994]. However, it might be possible to successfully train a model that outputs all curves at once using a Transformer [Vaswani et al., 2017] instead of an RNN as decoder. Note that this work only explored using a Transformer model as encoder, not as decoder. Egiazarian et al. [2020] use a Transformer as decoder, but still reduce the amount of curves output by the Transformer by splitting the input image into tiles and processing each tile independently.

An established practice to improve model performance when single models cannot be feasibly further improved is to use an ensemble of different models [Schapire, 1990, Ho, 1995]. Hence, it might be possible to improve results by using an ensemble of state-of-the-art prior work and this model.

One advantage of the developed method is its low runtime. This can be further reduced by parallelizing the iterative curve reconstruction algorithm as described in Line 16.

5.3.3 Further Tasks

As described in Sections 1.4, 4.1.2 and 5.1, an intrinsic characteristic of the developed method is that it is nearly end-to-end differentiable. While the focus of this work is on same-domain vectorization (i.e., turning raster clean animation frames into vector clean animation frames), in theory it is possible to extend the developed method to input images of different domains. For this, two parts of the method have to be adapted. Firstly, the marked curve reconstruction model is trained solely using clean line-art images as input and is thus not robust to input images of different domains. Due to the end-to-end differentiable nature of the model, this can be easily solved by training or finetuning the model on input images of different domains. Secondly, the curve identification algorithm needs to be adapted to input images of different domains, as laid out in Section 4.1.2. Alternatively, the algorithm could be extended by a learned model which converts a raster image of a different domain into a corresponding raster line-art image [Kugler, 2023]. This could then be used in combination to convert final animation frames in raster format into clean animation frames in vector format.

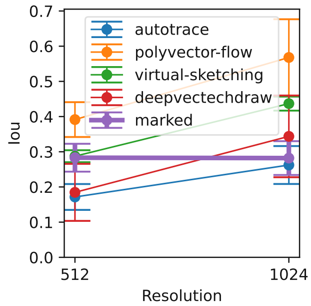
Furthermore, there exist other tasks to which the developed model could be extended. These include the generation of inbetween frames based on keyframes or clean animation frame colorization (see Section 2.2). Moreover, the model output could be constrained to exhibit temporal consistency, i.e., to consist of curves that remain consistent across frames of the same scene.

APPENDIX A

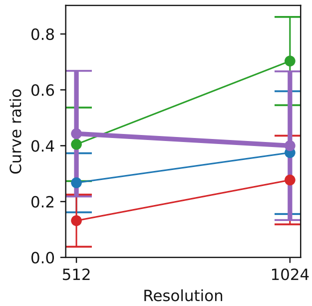
Additional Evaluation Results

This chapter includes figures showing additional evaluation results described in Section 4.3.4 and Section 4.3.5.

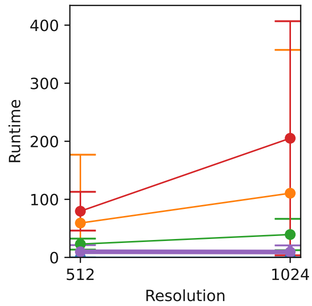
A.1 Quantitative Evaluation



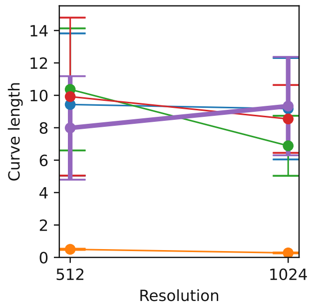
(a) The average IoU.



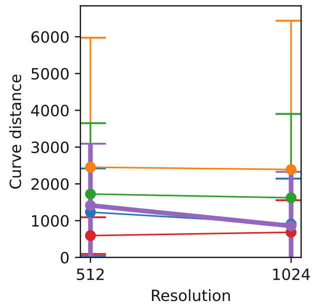
(b) The average curve ratio.



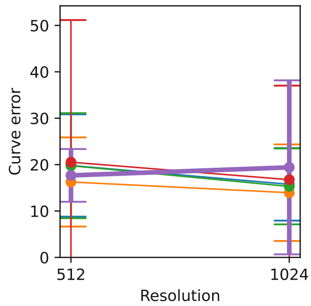
(c) The average runtime.



(d) The average curve length.

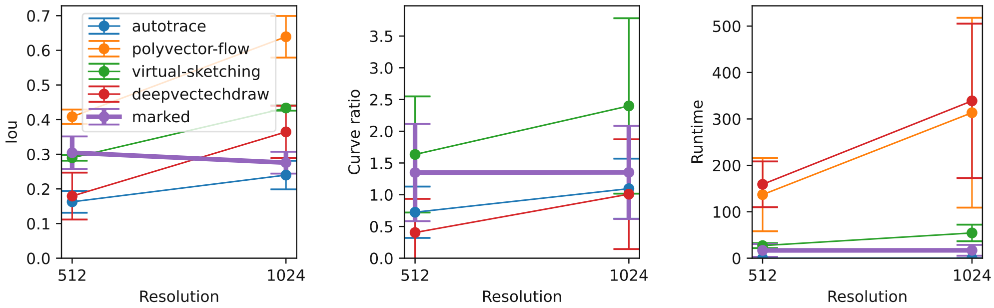


(e) The total curve distance.

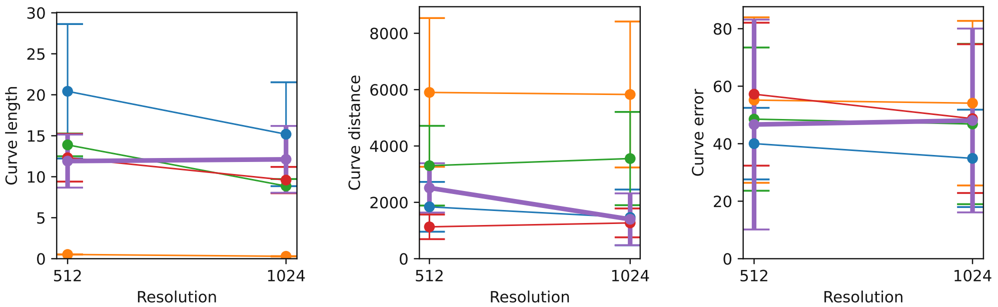


(f) The average curve error.

Figure A.1: Metrics for the line-art image vectorization methods evaluated on binarized images with 512px and 1024px resolution, respectively. Points denote the median of the metric, while vertical bars denote the 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 even if they are binarized.

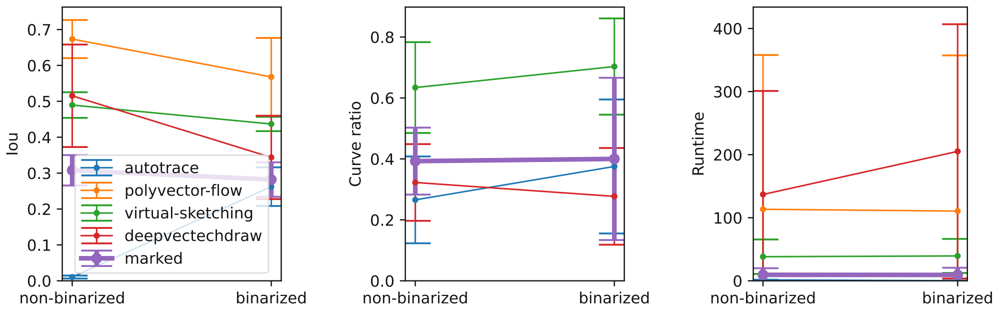


(a) The average IoU. (b) The average curve ratio. (c) The average runtime.

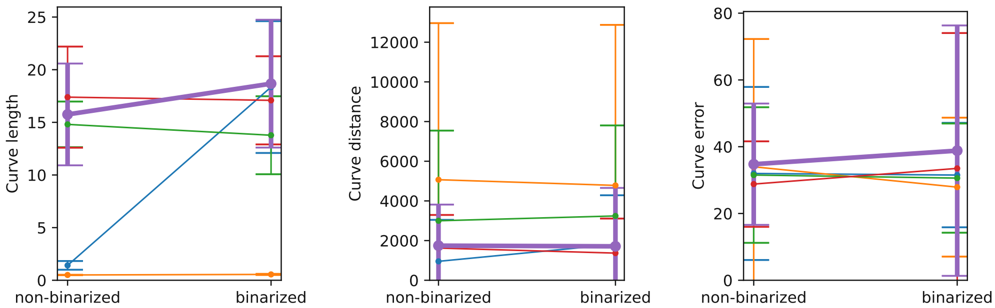


(d) The average curve length. (e) The total curve distance. (f) The average curve error.

Figure A.2: The same comparison as in Figure A.1 on the SketchBench test dataset instead of the Tonari test dataset.

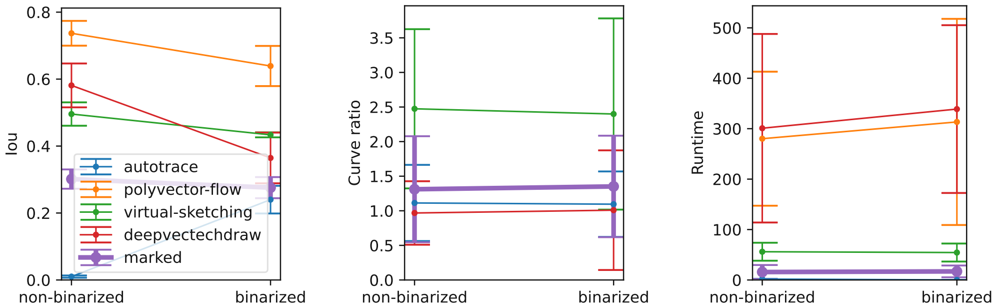


(a) The average IoU. (b) The average curve ratio. (c) The average runtime.

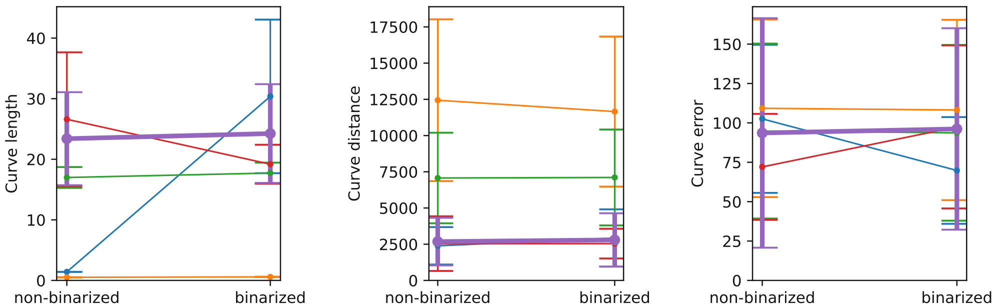


(d) The average curve length. (e) The total curve distance. (f) The average curve error.

Figure A.3: Metrics for the line-art image vectorization methods evaluated on binarized and non-binarized images with 1024px resolution, respectively. Points denote the median of the metric, while vertical bars denote the IQR. The metrics for the method developed in this work are emphasized. Note that they are not significantly affected by binarization even on high-resolution images.



(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 A.4: The same comparison as in Figure A.3 on the SketchBench test dataset instead of the Tonari test dataset.

A.2 Qualitative Evaluation

(a) Input raster image.

(b) Output of the developed method.

(c) Output of AutoTrace [Weber, 2002].

(d) Output of Egiazarian et al. [2020].

(e) Output of Puhachov et al. [2021].

(f) Output of Mo et al. [2021].

Figure A.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.

(a) Input raster image of the SketchBench dataset.

(b) Output of the developed method.

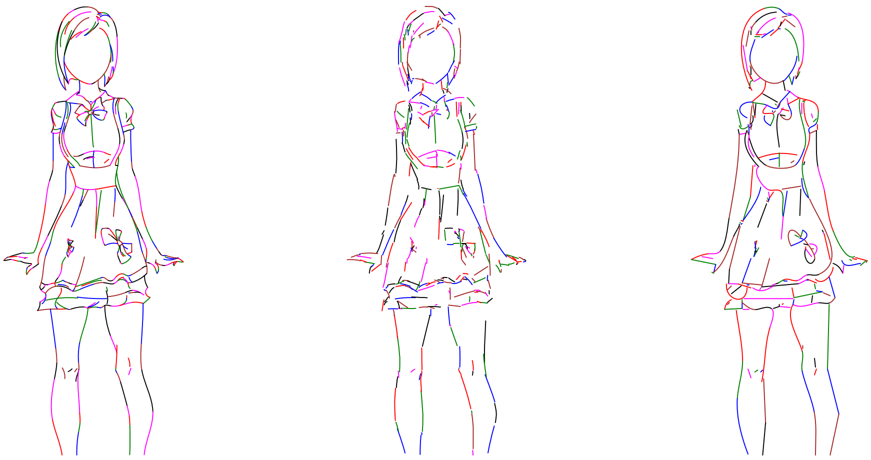
(c) Output of AutoTrace [Weber, 2002].

(d) Output of Egiazarian et al. [2020].

(e) Output of Puhachov et al. [2021].

(f) Output of the method by Mo et al. [2021].

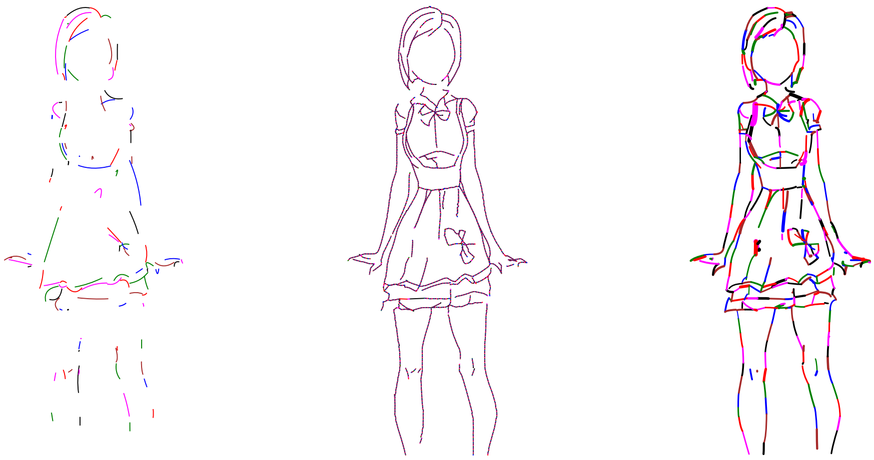
Figure A.6: The output vector image given a SketchBench professional sketch in raster format as input of each line-art image vectorization method studied in this work.



(a) Ground truth vector structure order.

(b) Output of the developed method.

(c) Output of AutoTrace [Weber, 2002].



(d) Output of Egiazarian et al. [2020].

(e) Output of Puhachov et al. [2021].

(f) Output of the method by Mo et al. [2021].

Figure A.7: The output vector image given a SketchBench professional sketch as input with the vector structure behind the images revealed by representing each curve with a mutually exclusive color.

List of Figures

List of Tables

List of Algorithms

Acronyms

AMP automatic mixed precision. 93

CNN Convolutional Neural Network. 20, 31, 32, 34–36, 39, 41, 42, 44, 46, 48, 49, 56–58, 66, 129, 137, 141, 145, 174

CPU central processing unit. 92, 94, 100

CUDA Compute Unified Device Architecture. 92

cuDNN CUDA Deep Neural Network. 92

FN false negative. 28

FP false positive. 28

GPGPU General-purpose computing on graphics processing units. 97

GPU Graphics Processing Unit. 24, 35, 46, 68, 81, 82, 92, 93, 97, 101, 104, 105, 151, 152, 173

GRU Gated Recurrent Unit. 45, 46

IoU Intersection-over-Union. 28, 29, 31, 68, 69, 97, 99, 101–117, 128, 137, 138, 140–147, 158–161, 167, 170, 171

IQR inter-quartile range. 78, 94, 101, 103, 104, 106, 108, 110–112, 114–116, 142, 144–147, 158, 160, 168, 169, 171

KL Kullback–Leibler. 51, 129, 137

LSTM Long Short-Term Memory. 45, 46

MAE mean absolute error. 27, 29, 30, 68, 98, 154

MiB mebibyte. 101, 104, 151, 173

MLP multi-layered perceptron. 20–22, 31, 34, 41, 44, 47–50, 66, 145, 166, 174

MSE mean squared error. 27, 29, 30, 68, 137, 154

NLP Natural Language Processing. 49

ONNX Open Neural Network Exchange. 92–94, 173

PNG Portable Network Graphics. 9, 88

ReLU Rectified Linear Unit. 21–24, 34, 66, 67, 129

RGB Red-Green-Blue. 7, 8, 31, 64, 65, 90, 129, 165

RNN Recurrent Neural Network. 20, 44–46, 48, 56–58, 129, 136, 154, 155

RO1 Research Objective 1. 5, 6, 55, 153

RO2 Research Objective 2. 5, 6, 153

RQ1 Research Question 1. 4, 5, 91, 97, 103, 149

SGD Stochastic Gradient Descent. 23, 29

SVG Scalable Vector Graphics. 9, 10, 71, 73, 76, 77, 87, 95, 96

TDNN Time Delay Neural Network. 39

TN true negative. 28

TP true positive. 28

VAE variational autoencoder. 51, 129, 131, 169

ViT Vision Transformer. 49

Bibliography

Shun-ichi Amari. A theory of adaptive pattern classifiers. *IEEE Trans. Electron. Comput.*, 16(3):299–307, 1967. doi: 10.1109/PGEC.1967.264666. URL `https://doi.org/10.1109/PGEC.1967.264666`.

Inc. Anaconda. Anaconda Software Distribution, 2020. URL `https://docs.anaconda.com/`.

Rohan Anil, Vineet Gupta, Tomer Koren, and Yoram Singer. Memory efficient adaptive optimization. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9746–9755, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/8f1fa0193ca2b5d2fa0695827d8270e9-Abstract.html`.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL `http://arxiv.org/abs/1607.06450`.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *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/1409.0473`.

Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. `https://github.com/onnx/onnx`, 2019.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv e-prints*, art. arXiv:1803.01271, Mar 2018.

Amelia Bellamy-Royds, Bogdan Brinza, Chris Lilley, Dirk Schulze, David Storey, Eric Willigers, David Dailey, Eric Eastwood, Jarek Foksa, Daniel Holbert, Paul LeBeau, Robert Longson, Henri Manson, Ms2ger, Kari Pihkala, Philip Rogers, David Zbarsky, Patrick Dengler, Jon Ferraiolo, Anthony Grasso, Dean Jackson, and Fujisawa Jun.

Scalable vector graphics (SVG) 2. Candidate recommendation, W3C, October 2018. URL https://www.w3.org/TR/2018/CR-SVG2-20181004/.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994. ISSN 1045-9227. doi: 10.1109/72.279181. URL https://doi.org/10.1109/72.279181.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

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.

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. URL https://openaccess.thecvf.com/content/CVPR2021/html/Bhunia_Vectorization_and_Rasterization_Self-Supervised_Learning_for_Sketch_and_Handwriting_CVPR_2021_paper.html.

Léon Bottou, Françoise Fogelman-Soulié, Pascal Blanchet, and Jean-Sylvain Liénard. Experiments with time delay networks and dynamic time warping for speaker independent isolated digits recognition. In *First European Conference on Speech Communication and Technology, EUROSPEECH 1989, Paris, France, September 27-29, 1989*, pages 2537–2540. ISCA, 1989. URL http://www.isca-speech.org/archive/eurospeech_1989/e89_2537.html.

T. Boutell. Png (portable network graphics) specification version 1.0. RFC 2083, RFC Editor, March 1997.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1151. URL https://www.aclweb.org/anthology/D17-1151.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon

Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html`.

CACANi Pte Ltd. Cacani sample animation files, 2022. URL `https://cacani.sg/sample-files/?v=f003c44deab6`. accessed 2023-01-05.

Alexandre S. D. Celibidache and P.A. WORKS. Sakura quest, 2017. URL `http://sakura-quest.com/caststaff/`. [Online; accessed 2023-08-14].

CELSYS, Inc. Clip Studio, 2021. URL `https://www.clipstudio.net/en/`.

Kumar Chellapilla, Sidd Puri, and Patrice Simard. High Performance Convolutional Neural Networks for Document Processing. In Guy Lorette, editor, *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France), October 2006. Université de Rennes 1, Suvisoft. URL `https://inria.hal.science/inria-00112631`. http://www.suvisoft.com.

Ke Chen, V. Kvasnicka, P. C. Kanen, and Simon Haykin. Multi-valued and universal binary neurons: Theory, learning, and applications [book review]. *IEEE Trans. Neural Networks*, 12(3):647, 2001. doi: 10.1109/TNN.2001.925572. URL `https://doi.org/10.1109/TNN.2001.925572`.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. URL `http://arxiv.org/abs/1410.0759`.

Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014a.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In Dekai Wu, Marine Carpuat, Xavier Carreras, and Eva Maria Vecchi, editors, *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111. Association for Computational Linguistics, 2014b. doi: 10.3115/v1/W14-4012. URL `https://aclanthology.org/W14-4012/`.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 3642–3649. IEEE Computer Society, 2012. doi: 10.1109/CVPR.2012.6248110. URL https://doi.org/10.1109/CVPR.2012.6248110.

Alex Clark. Pillow (pil fork) documentation, 2015. URL https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, 2011. doi: 10.5555/1953048.2078186. URL https://dl.acm.org/doi/10.5555/1953048.2078186.

CourtBouillon. Cairosvg, March 2021. URL https://cairosvg.org/documentation/. accessed on 2022-05-04.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control. Signals Syst.*, 2(4):303–314, 1989. doi: 10.1007/BF02551274. URL https://doi.org/10.1007/BF02551274.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *NeurIPS*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

P. de Faget de Casteljau. *Formes à pôles*. Mathematiques et CAO. Hermes, Paris, 1986. ISBN 9782866010423.

Rina Dechter. Learning while searching in constraint-satisfaction-problems. In Tom Kehler, editor, *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science*, pages 178–185. Morgan Kaufmann, 1986. URL http://www.aaai.org/Library/AAAI/1986/aaai86-029.php.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019,* *Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423. URL https://doi.org/10.18653/v1/n19-1423.

Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945. doi: https://doi.org/10.2307/1932409. URL https://esajournals.onlinelibrary.wiley.com/doi/abs/10.2307/1932409.

Tor Dokken, Morten Dæhlen, Tom Lyche, and Knut Mørken. Good approximation of circles by curvature-continuous bézier curves. *Computer Aided Geometric Design*, 7(1):33–41, 1990. ISSN 0167-8396. doi: https://doi.org/10.1016/0167-8396(90)90019-N. URL https://www.sciencedirect.com/science/article/pii/016783969090019N.

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(2):295–307, 2016. doi: 10.1109/TPAMI.2015.2439281. URL https://doi.org/10.1109/TPAMI.2015.2439281.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Stuart Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, aug 1962. doi: 10.1016/0022-247x(62)90004-5. URL https://doi.org/10.1016%2F0022-247x%2862%2990004-5.

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016.

DWANGO Co., Ltd. Opentoonz sample data, 2023. URL https://cacani.sg/sample-files/?v=f003c44deab6. accessed 2023-01-05.

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.

Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4):44:1–44:10, 2012.

Kunihiko Fukushima and Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognit.*, 15(6): 455–469, 1982. doi: 10.1016/0031-3203(82)90024-3. URL https://doi.org/10.1016/0031-3203(82)90024-3.

Furansujin Connection. Les étapes de fabrication d'un anime, 2016. URL https://web.archive.org/web/20220401075414/http://www.furansujinconnection.com/les-etapes-de-fabrication/. accessed on 2022-04-01.

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.

Grove Karl Gilbert. Finley's tornado predictions. *American Meteorological Journal*, 1(5): 166–172, September 1884.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, Oct 2017. ISSN 2162-237X. doi: 10.1109/TNNLS.2016.2582924.

Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Frédo Durand, and Adrien Bousseau. Opensketch: A richly-annotated dataset of product design sketches. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 38, 11 2019.

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.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com.

David Ha and Douglas Eck. A neural representation of sketch drawings. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Hy6GHpkCW.

Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1225–1234. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/hardt16.html.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.123. URL http://dx.doi.org/10.1109/ICCV.2015.123.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016a. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. 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 IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 630–645. Springer, 2016b. doi: 10.1007/978-3-319-46493-0_38. URL https://doi.org/10.1007/978-3-319-46493-0_38.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. doi: 10.1126/science.1127647. URL https://www.science.org/doi/abs/10.1126/science.1127647.

Geoffrey Hinton, Nitish Srivatsava, and Kevin Swersky. Lecture 6e - rmsprop: Divide the gradient by a running average of its recent magnitude. In *Neural Networks for Ma-*

chine Learning, 2014. URL http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995. doi: 10.1109/ICDAR.1995.598994.

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Master's thesis, Technical University of Munich, 1991.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. doi: 10.1016/0893-6080(89)90020-8. URL https://doi.org/10.1016/0893-6080(89)90020-8.

Benran Hu. Yet-Another-Anime-Segmenter, 2020. URL https://github.com/zymk9/Yet-Another-Anime-Segmenter/.

Zhewei Huang, Shuchang Zhou, and Wen Heng. Learning to paint with model-based deep reinforcement learning. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 8708–8717. IEEE, 2019. doi: 10.1109/ICCV.2019.00880. URL https://doi.org/10.1109/ICCV.2019.00880.

Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73 – 101, 1964. doi: 10.1214/aoms/1177703732. URL https://doi.org/10.1214/aoms/1177703732.

J. D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

Ivan Huska. Elephant, 2009. URL https://www.easy-drawings-and-sketches.com/. [Online; accessed 2023-08-14].

ImageMagick Studio LLC. ImageMagick, January 2023. URL https://imagemagick.org.

Inkscape Project. Inkscape, 2020. URL https://inkscape.org.

Institute of Electrical and Electronics Engineers. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019. doi: 10.1109/IEEESTD.2019.8766229.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/ioffe15.html`.

Paul Jaccard. The distribution of the flora in the alpine zone.1. *New Phytologist*, 11(2):37–50, 1912. doi: https://doi.org/10.1111/j.1469-8137.1912.tb05611.x. URL `https://nph.onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.1912.tb05611.x`.

Glenn Jocher. Yolov5 by ultralytics, 2020. URL `https://github.com/ultralytics/yolov5`.

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`.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2342–2350, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/jozefowicz15.html`.

David Keppel, David MacKenzie, and Assaf Gordon. GNU Time, 2018. URL `https://www.gnu.org/software/time/`.

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`.

Jonas Moritz Kohler, Hadi Daneshmand, Aurélien Lucchi, Thomas Hofmann, Ming Zhou, and Klaus Neymeyr. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 806–815. PMLR, 2019. URL `http://proceedings.mlr.press/v89/kohler19a.html`.

J. F. Kolen and S. C. Kremer. *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, pages 237–243. 2001. doi: 10.1109/9780470544037.ch14.

Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991. doi: https://doi.org/10.1002/aic.690370209. URL https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690370209.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012. URL https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

Felix Kugler. Reconstructing production data from drawn limited animation. Master's thesis, Research Unit of Computer Graphics, Institute of Visual Computing and Human-Centered Technology, Faculty of Informatics, TU Wien, Favoritenstrasse 9-11/E193-02, A-1040 Vienna, Austria, September 2023. URL https://www.cg.tuwien.ac.at/research/publications/2023/Kugler-2021/.

S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951. doi: 10.1214/aoms/1177729694. URL https://doi.org/10.1214/aoms/1177729694.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 12 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.4.541. URL https://doi.org/10.1162/neco.1989.1.4.541.

Jerry Li. Pixiv dataset, 2017. URL https://github.com/jerryli27/pixiv_dataset. accessed 2022-04-05.

Mengtian Li, Zhe Lin, Radomír Mˇ ech, Ersin Yumer, and Deva Ramanan. Photo-sketching: Inferring contour drawings from images. In *WACV*, 2019.

Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 39(6):193:1–193:15, 2020.

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.

Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2999–3007. IEEE Computer Society, 2017. doi: 10.1109/ICCV.2017.324. URL https://doi.org/10.1109/ICCV.2017.324.

Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, University of Helsinki, 1970.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 5747–5763. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.463. URL https://doi.org/10.18653/v1/2020.emnlp-main.463.

Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9628–9639, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/60106888f8977b71e1f15db7bc9a88d1-Abstract.html.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 11966–11976. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01167. URL https://doi.org/10.1109/CVPR52688.2022.01167.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP*

2015, Lisbon, Portugal, September 17-21, 2015, pages 1412–1421. The Association for Computational Linguistics, 2015. doi: 10.18653/v1/d15-1166. URL https://doi.org/10.18653/v1/d15-1166.

H. B. Mann and D. R. Whitney. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1): 50 – 60, 1947. doi: 10.1214/aoms/1177730491. URL https://doi.org/10.1214/aoms/1177730491.

Hiromichi Masuda, Tadashi Sudo, Toshiyuki Koudate, Atsushi Matsumoto, Kazuo Rikukawa, Tomotaka Ishida, Yasuo Kameyama, Yuji Mori, and Masahiro Hasegawa. Anime industry report 2022 summary, March 2023. URL https://aja.gr.jp/english/japan-anime-data.

Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), mar 2014. ISSN 1075-3583.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=r1gs9JgRZ.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL http://arxiv.org/abs/1301.3781.

Tomáš Mikolov. *STATISTICAL LANGUAGE MODELS BASED ON NEURAL NETWORKS*. Ph.d. thesis, Brno University of Technology, Faculty of Information Technology, 2012. URL https://www.fit.vut.cz/study/phd-thesis/283/.

Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *Fourth International Conference on 3D Vision, 3DV 2016, Stanford, CA, USA, October 25-28, 2016*, pages 565–571. IEEE Computer Society, 2016. doi: 10.1109/3DV.2016.79. URL https://doi.org/10.1109/3DV.2016.79.

Marvin Minsky and Seymour Papert. *Perceptrons. An Introduction to Computational Geometry.* MIT Press, 1969.

Haoran Mo, Edgar Simo-Serra, Chengying Gao, Changqing Zou, and Ruomei Wang. General virtual sketching framework for vector line art. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2021)*, 40(4):51:1–51:14, 2021.

Motion Picture Producers Association of Japan, Inc. Movies with box office gross receiopts exceeding 1 billion yen, 2021. URL http://www.eiren.org/boxoffice_e/2021.html. [Online; accessed 2023-08-14].

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL http://dl.acm.org/citation.cfm?id=3104322.3104425.

S.J. Napier. *Anime from Akira to Howl's Moving Castle: Experiencing Contemporary Japanese Animation*. St. Martin's Publishing Group, 2016. ISBN 9781250117724. URL https://books.google.com/books?id=UnuLCwAAQBAJ.

Gioacchino Noris, Alexander Hornung, Robert W. Sumner, Maryann Simmons, and Markus Gross. Topology-driven vectorization of clean line drawings. *ACM Trans. Graph.*, 32(1), feb 2013. ISSN 0730-0301. doi: 10.1145/2421636.2421640. URL https://doi.org/10.1145/2421636.2421640.

NVIDIA Corporation. Cuda, release: 10.2.89, 2020. URL https://developer.nvidia.com/cuda-toolkit.

László Németh. 30-degree color wheel with rgb pixels of the colors. Wikimedia Commons, June 2013. URL https://commons.wikimedia.org/wiki/File:RGB_color_wheel_pixel_30.svg. accessed 2023-05-06.

David Olsen. svgelements, 2022. URL https://github.com/meerk40t/svgelements.

ONNX Runtime developers. Onnx runtime. https://onnxruntime.ai/, 2021. Version: 1.15.1.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *CoRR*, abs/2203.02155, 2022. doi: 10.48550/arXiv.2203.02155. URL https://doi.org/10.48550/arXiv.2203.02155.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith

Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Philipp Petersen and Felix Voigtländer. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330, 2018. doi: 10.1016/j.neunet.2018.08.019. URL https://doi.org/10.1016/j.neunet.2018.08.019.

Ivan Puhachov, William Neveu, Edward Chien, and Mikhail Bessmeltsev. Keypoint-driven line drawing vectorization via polyvector flow. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 40(6), December 2021. doi: 10.1145/3478513.3480529.

Python Core Team. *Python: A dynamic, open source programming language*. Python Software Foundation, 2019. URL https://www.python.org/. Python version 3.8.12.

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.

A. J. Robinson and Frank Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, Cambridge, UK, 1987.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 10674–10685. IEEE, 2022. doi: 10.1109/CVPR52688.2022.01042. URL https://doi.org/10.1109/CVPR52688.2022.01042.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells III, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015. doi: 10.1007/978-3-319-24574-4_28. URL https://doi.org/10.1007/978-3-319-24574-4_28.

F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, pages 65–386, 1958.

Andy Roth. svgpathtools, 2021. URL https://github.com/mathandy/svgpathtools.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL https://doi.org/10.1038/323533a0.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Labelme: A database and web-based tool for image annotation. *Int. J. Comput. Vis.*, 77(1-3): 157–173, 2008. doi: 10.1007/s11263-007-0090-8. URL https://doi.org/10.1007/s11263-007-0090-8.

Itay Safran and Ohad Shamir. Depth-width tradeoffs in approximating natural functions with neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2979–2987. PMLR, 2017. URL http://proceedings.mlr.press/v70/safran17a.html.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/905056c1ac1dad141560467e0a99e1cf-Paper.pdf.

Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, Jun 1990. ISSN 1573-0565. doi: 10.1007/BF00116037. URL https://doi.org/10.1007/BF00116037.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Peter Selinger. Potrace: a polygon-based tracing algorithm, 2003.

S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples)†. *Biometrika*, 52(3-4):591–611, 12 1965. ISSN 0006-3444. doi: 10.1093/biomet/52.3-4.591. URL https://doi.org/10.1093/biomet/52.3-4.591.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016. doi: 10.1038/nature16961. URL `https://doi.org/10.1038/nature16961`.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2256–2265. JMLR.org, 2015. URL `http://proceedings.mlr.press/v37/sohl-dickstein15.html`.

T. Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and ist application to analyses of the vegetation on danish commons. *Det Kongelige Danske Videnskabernes Selskab*, 5(4):1–34, 1948.

X. Soria, E. Riba, and A. Sappa. Dense extreme inception network: Towards a robust cnn model for edge detection. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1912–1921, Los Alamitos, CA, USA, mar 2020. IEEE Computer Society. doi: 10.1109/WACV45572.2020.9093290. URL `https://doi.ieeecomputersociety.org/10.1109/WACV45572.2020.9093290`.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2377–2385, 2015. URL `https://proceedings.neurips.cc/paper/2015/hash/215a71a12769b056c3c32e7299f1c5ed-Abstract.html`.

Bolan Su, Shijian Lu, and Chew Lim Tan. Binarization of historical document images using the local maximum and minimum. In David S. Doermann, Venu Govindaraju, Daniel P. Lopresti, and Premkumar Natarajan, editors, *The Ninth IAPR International Workshop on Document Analysis Systems, DAS 2010, June 9-11, 2010, Boston, Massachusetts, USA*, ACM International Conference Proceeding Series, pages 159–166. ACM, 2010. doi: 10.1145/1815330.1815351. URL `https://doi.org/10.1145/1815330.1815351`.

Hao Su, Jianwei Niu, Xuefeng Liu, Jiahe Cui, and Ji Wan. Vectorization of raster manga by deep reinforcement learning. *CoRR*, abs/2110.04830, 2021. URL `https://arxiv.org/abs/2110.04830`.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013. URL http://dl.acm.org/citation.cfm?id=3042817.3043064.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9. IEEE Computer Society, 2015. doi: 10.1109/CVPR.2015.7298594. URL https://doi.org/10.1109/CVPR.2015.7298594.

Ole Tange. GNU Parallel 20220922 ('Elizabeth') released, September 2022. URL https://doi.org/10.5281/zenodo.7105792.

T.T. Tanimoto. An elementary mathematical theory of classification and prediction. Technical report, International Business Machines Corporation, 1958.

Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rJiNwv9gg.

TorchVision maintainers and contributors. TorchVision: PyTorch's Computer Vision library, November 2016. URL https://github.com/pytorch/vision.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023. doi: 10.48550/arXiv.2302.13971. URL https://doi.org/10.48550/arXiv.2302.13971.

Takeru Uchida, Shōsetsuka ni Narō, and Isekai Cheat Magician Production Committee. Isekai cheat magican, 2019. URL http://isekai-cheat-magician.com/staff_cast/. [Online; accessed 2023-08-14].

Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in python. *PeerJ*, 2:e453, June 2014. ISSN 2167-8359. doi: 10.7717/peerj.453. URL https://doi.org/10.7717/peerj.453.

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.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 2020. doi: https://doi.org/10.1038/s41592-019-0686-2.

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.

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

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6): 80–83, 1945. ISSN 00994987. URL http://www.jstor.org/stable/3001968.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4148–4158. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7003-the-marginal-value-of-adaptive-gradient-methods-in-machine-learning.pdf.

Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5987–5995. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.634. URL https://doi.org/10.1109/CVPR.2017.634.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/xuc15.html.

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.

Ge Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 17084–17097, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/8df7c2e3c3c3be098ef7b382bd2c37ba-Abstract.html.

Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. A mean field theory of batch normalization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=SyMDXnCcF7.

Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative Study of CNN and RNN for Natural Language Processing. *arXiv e-prints*, art. arXiv:1702.01923, Feb 2017.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press, 2016. URL http://www.bmva.org/bmvc/2016/papers/paper087/index.html.

Lvim Zhang. sketchkeras, 2017. URL https://github.com/lllyasviel/sketchKeras. accessed 2023-04-18.

Lvmin Zhang, Xinrui Wang, Qingnan Fan, Yi Ji, and Chunping Liu. Generating manga from illustrations via mimicking manga creation workflow. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

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.

Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Stat. Anal. Data Min.*, 5(5):363–387, 2012. doi: 10.1002/sam.11161. URL `https://doi.org/10.1002/sam.11161`.