A Deep Learning-Based Perception-Driven Vectorization Approach For Semi-Structured Imagery

A Special Problem by

Sean Francis N. Ballais
2015-04562
BS CS

Presented to the Faculty of the
Division of Natural Sciences and Mathematics

In Partial Fulfillment of the Requirements
For the Degree of
Bachelor of Science in Computer Science

University of the Philippines Visayas
TACLOBAN COLLEGE
Tacloban City

Month Year

This special problem, entitled "**A DEEP LEARNING-BASED PERCEPTION-DRIVEN VECTORIZATION APPROACH FOR SEMI-STRUCTURED IMAGERY**", prepared and submitted by **SEAN FRANCIS N. BALLAIS**, in partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE** is hereby accepted.

PROF. VICTOR M. ROMERO II
Special Problem Adviser

Accepted as partial fulfillment of the requirements for the degree of **BACHELOR OF SCIENCE IN COMPUTER SCIENCE**.

DR. EULITO V. CASAS JR.
Chair, DNSM

Acknowledgements

First of all I would like to thank the Lord for his guidance during the course of my research and for making this thesis possible. I would like to thank my family who served as my inspiration, and never failed to support me all throughout my studies. Thanks to ...

Abstract

Although the abstract is the first thing that appears in the thesis, it is best written last after you have written your conclusion. It should contain spell out your thesis problem and describe your solution clearly.

Make sure your abstract fits in one page !

Table of Contents

Acknowledgements **iii**

Abstract **iv**

Table of Contents **vi**

List of Tables **vii**

List of Figures **viii**

1 Introduction **1**
1.1 The Problems of Raster Graphics 3
1.2 Vectorization To The Rescue 5
1.3 B-Splines 8
B-Spline Approximation 9

2 Review of Related Literature **12**
2.1 Works Based on Great Deluge 12

3 Statement of the Problem **15**

4 Objectives **16**

5 Proposed Methodology **17**
5.1 Image Preprocessing 17
Corner Detection 17
Region Segmentation 18
5.2 Curve Approximation 19
Point Sequence Generation 19
Point Sequence Curve Approximation 20

5.3 Region Colouring and Merging 27

6 Describing How You Validated Your Approach. 29

7 Stating Your Results and Drawing Insights From Them. 30

8 Summarizing Your Thesis and Drawing Your Conclusions. 31

A What should be in the Appendix 32

Bibliography 33

List of Tables

List of Figures

1.1 When zoomed in enough, each individual pixel of a raster image is visible. Meme image obtained from http://thesismemes.tumblr.com/post/73483120281. 2

1.2 Different interpolation algorithms will produce different results. As seen in the image, the quality will also differ, from an image looking blocky to an image looking blurry. Cat meme image obtained from https://www.hercampus.com/school/uwindsor/school-thoughts-told-grumpy-cat-memes.

1.3 Zooming in closely at the same image, but one being in raster format (top right) and the other in vector (though hand-drawn; bottom left), quickly reveals the quality differences of both image formats. Raster images will show you individual pixels when close enough, but vectors will remain smooth. 6

5.1 The network architecture of the Point Parametrization Network (PPN) 22

5.2 The network architecture of the Knot Selection Network (PPN) . . . 25

Chapter 1

Introduction

In computer graphics, most images are typically stored as a sequence of dots in a rectangular grid (see Fig. 1.1). Each dot is called a pixel, a small part of an image that holds one specific colour, Photographs, also called natural images [13], are one of, if not the most, common images that are stored in this manner. Many digital forms of art or any graphics work, such as paintings, posters, and icons, are also stored the same. Digital images stored in this manner are called *raster images*. These images are stored in various image formats. The most commonly used formats are JPEG, GIF, BMP, TIFF, and PNG. Each have their pros and cons, from quality of the resulting image to the file size. Nevertheless, they all still accomplish the task of holding raster image data. Everything you see in the displays of devices such as laptops and mobile devices is a raster image. Computer displays are collections of pixels, in the common definition of a dot on the screen, which computers map images to to be able display them. This is the reason why **all** *displayed* images are raster images.

A positive aspect of raster images is their simplicity. As mentioned earlier, raster images consists of a grid of pixels (also called a pixel matrix in other literature [25]). This pixel grid can simply be assigned a combination of colour values to create an

Figure 1.1: When zoomed in enough, each individual pixel of a raster image is visible. Meme image obtained from `http://thesismemes.tumblr.com/post/73483120281`.

image. As such, working with raster images can be analogous to painting in the real world [4]. Given the right combinations of colours, we can produce natural images, i.e. photographs [13]. Intuitively, this means that we can store fine details in a raster image [21]. This is in contrast to *vector images*, which use a series of points and mathematical calculations to form lines and shapes. Vector images are unable to display lush colour depth and keep granularity, as found in raster images, as they use solid colours or gradients [4][5]. There are studies that have been conducted in improving and utilizing *gradient meshes*, a vector graphics primitive that allows for intricate colour gradients in regular quadrilateral meshes first introduced by Adobe Illustrator, to produce photorealistic vector images. However, as noted in the paper by Jian, S, Liang, L., Wen, F., and Shum, H., simple gradient meshes are insufficient to keep the fine details of images [6][21]. It is also important to mention that vector graphics, despite represented as mathematical calculations, are still converted to raster format in a process called *rasterization* for it to be displayed on-screen, since many modern screens are raster displays [22].

1.1 The Problems of Raster Graphics

With all the pros raster graphics have, it does not mean raster graphics are not without their caveats. Raster graphics have their own disadvantages which could affect the image quality and their use.

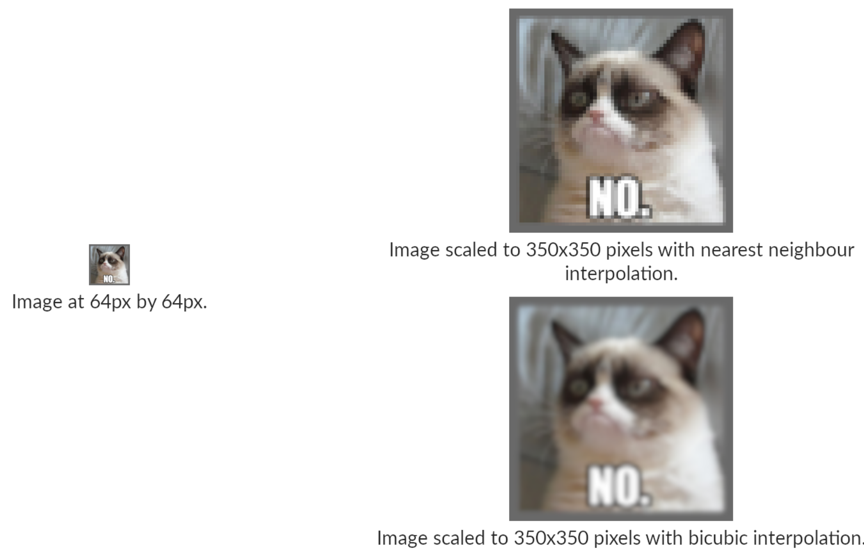

Figure 1.2: Different interpolation algorithms will produce different results. As seen in the image, the quality will also differ, from an image looking blocky to an image looking blurry. Cat meme image obtained from `https://www.hercampus.com/school/uwindsor/school-thoughts-told-grumpy-cat-memes`.

Raster graphics are **resolution-dependent**. This simply means that raster images are in their highest quality in the resolution they are initially created in and attempting to scale it up will gradually degrade the quality of the image as the image size or resolution grows larger. Rasters only have a finite number of pixels. Increasing the size of an image (also called upsampling [13] or single-image super resolution [27]) would entail moving the individual pixels into different locations depending on the scaling factor, the distance the individual pixels will be moved to horizontally and vertically. Upsampling will create empty pixels in between the shifted pixels when there is nothing done to substitute the empty pixels. This will create an unusable image [10]. We can utilize interpolation to fill these empty pixels with colour. *Classical* image upsampling approaches approximate colour and intensity values of these empty pixels are calculated based on the values of surrounding pixels, typically the shifted pixels. However, the specifics are dependent on the interpolation algorithm used in upscaling the images. Commonly used classical interpolation methods for resizing images include Nearest-Neighbour, Linear Interpolation, and Cubic Interpolation, which most, if not all, are readily available in popular raster image editing applications, such as Adobe Photoshop [2] and GIMP [1]. The results produced by these interpolation algorithms typically suffer from blurring of sharp edges and ringing artifacts due to the fact that the algorithms do not assume anything about the data [14][19]. See Fig. 1.2 for an example of blurring caused by scaling. There are adaptive image scaling techniques that consider image features such as edge information and texture to scale images with better quality than the classical methods. Examples of adaptive techniques are content-aware image resizing, seam curving, and warping-based methods. These techniques have their downsides as they take more computational time than their non-adaptive counterparts and may produce unexpected or even unsatisfactory results [19]. One of the latest advancements in upscaling images involves the use of artifical intelligence, specifically *neural networks*, as found in the works of AI Gigapixel and by Yang, C. Ma, C. and Yang, M.. They produce high quality upscaled images and is a significant improvement over previous non-AI based upscaling techniques.

However, they require high-end expensive hardware to produce results in the shortest amount of time possible. In the case of AI Gigapixel, a laptop with an integrated graphics card takes 20 minutes to produce a final high resolution image. For the work by Yang. C, Ma.C., and Yang. M., they utilized an Nvidia Titan Xp, a high-end GPU that costs $1,200 as of November 18, 2018 [3], to upscale a 520x520px image 2x, 4x, and 8x its size, and took 0.8s, 2.1s, and 4.4s, respectively, to complete [26][23].

1.2 Vectorization To The Rescue

Vectorization is the process of converting raster images into vector images [28]. Vector graphics uses collections of geometric primitives, such as points, curves, and points, and mathematical calculations to form an image [4]. Unlike raster graphics which uses a large pixel matrix (which will require large spaces without using proper image compression), vector graphics are able to smoothly scale to different resolutions, large or small, without any degradation in image quality [25][6]. This makes them **resolution-independent**. Vector graphics innately have this property due to their reliance on mathematics, instead of context-free pixel grids. Each primitive have their own mathematic formulas which, obviously, stay the same no matter what the size of an image is. As such, the primitives can simply be re-rendered whenever the image is scaled [4][5]. Vector graphics also allow for easier editing [13][21], as you only need to modify individual polygons, lines, and curves, instead of dealing with individual pixels like you normally would when using raster images editors.

Vectorization would often be done manually. In a study by Hoshyari, et. al. [13], each of their raster images, which includes icons and small graphic illustrations, take 30-45 minutes to be vectorized by an artist. More than 7 million man hours are being

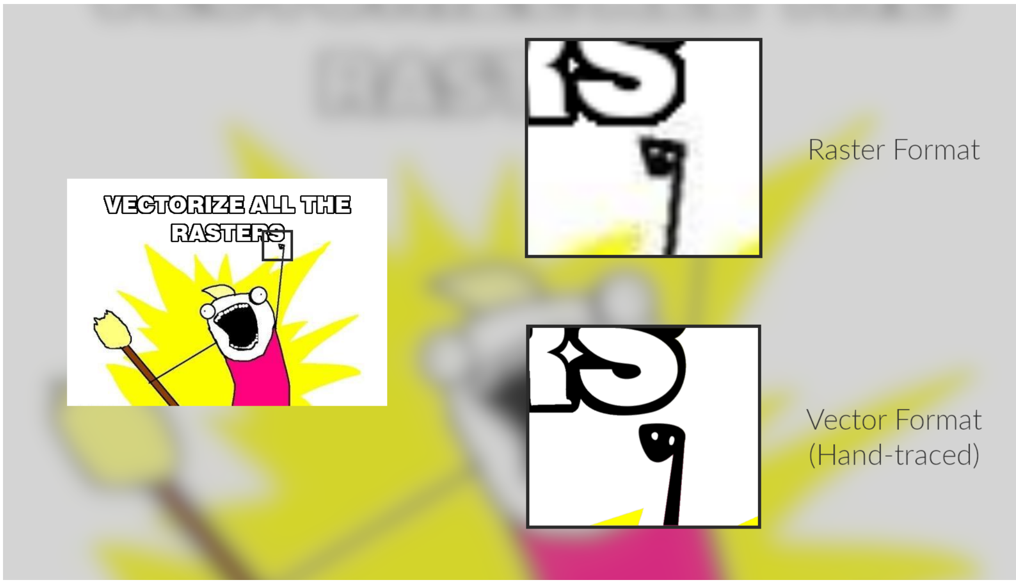

Figure 1.3: Zooming in closely at the same image, but one being in raster format (top right) and the other in vector (though hand-drawn; bottom left), quickly reveals the quality differences of both image formats. Raster images will show you individual pixels when close enough, but vectors will remain smooth.

.

spent on vectorizing raster graphics in the United States every year, according to a survey in the PhD dissertation of J.R. Diebel entitled, "Bayesian image vectorization: The probabilistic inversion of vector image rasterization" [28]. Demand is, therefore, there for a robust vectorization algorithm.

Vectorization have been applied in many cases including, but not limited to, 2D maps and natural images. There are multiple methods that can be utilized in the vectorization of raster images. Their results differ from one method to another and even from one input to another, as many vectorization methods are fine tuned to specific inputs, such as those by Hoshyari, S., et. al. (semi-structured images) [13], Kopf, J. and Lischinski, D. (pixel art) [14], and Bessmeltsev, M. and Solomon, J. (line drawings) [7]. Additionally, the ability to utilize GPUs for computational tasks has allowed parallelization of vectorization such as in the paper where a GPU was used to vectorize a video stream in real time [25].

Many vectorization methods target natural images. As noted by Hoshyari, S., et. al. many of these natural image vectorization methods utilize *image segmentation* to identify the portions of the image that will be converted into geometric primitives. These primitives are then filled with solid colors (e.g. such as what was done by Birdal, T. and Bala, E. [8]), gradient meshes (which are used in Adobe Illustrator and Corel CorelDraw [21][6]), and/or diffusion curves (such was the case in the paper by Xie, G. Sun, X., Tong, X., and Nowrouzezahrat, D. [24]). These vectorization methods produce differing results whose image qualities vary. Some results are as close as possible to the original raster image. Typically, gradient meshes and diffusion curves were utilized to achieve these results [24][21][6]. Others produce results with obvious colour segmentations, as seen in results that purely utilize solid colours (see [8] for an example).

Natural images are not the only raster images that are being vectorized. Images called semi-structured images are also candidates for image vectorization. *Semi-structured images* (SSIs) are images that consists of distinctly coloured regions and have well-defined boundaries [13]. Logos, cartoons, clip art, computer icons, and even simple graphical illustrations (such as flat 2D art) can considered to be semi-structured images. 60% of the 10 million images to be vectorized are semi-structured images [28]. This makes the demand for vectorizing these types of images evident. Various methods for vectorizing semi-structured images have been proposed by numerous papers. The common methodology of these proposals is that they attempt to fit curves or Bezier splines on the boundaries of each region in SSIs and fill in the appropriate colour. However, each of these methods naturally have their own unique ways of fitting curves. In the work of Kopf, J., and Lischinski, D. [14], they use similarity graphs and additional intermediate steps in identifying the regions of an image, though their work is targeted at pixel art. Another paper by Yang, M., et. al. would directly optimize the shapes of individual Bezier segments connecting each boundary transition vertices to produce high fidelity vectorized images [13][28]. The work by Hoshyari, S., et. al. can be seen as a complement to the aforementioned paper. Their work utilizes human perceptual cues, primarily guided by Gestalt psychology, to produce vectorizations of semi-structured images that align much more closely to what viewers expect from a raster image [13].

An alternative method we can perform for image vectorization is to use machine learning via convolutional neural networks, a type of neural network that is well suited for image classification for their ability to learn various image features [12], to create vectorizations of semi-structured images by having the computer learn to produce Bezier curves of image region boundaries that align well with the expected vectorization. This is the method that this paper is proposing. A similar work, in that convolutional neural networks were for vectorization, to this is that of the paper of Simo-Serra, E., Iizuka, S., Sasaki, K., and Ishikawa, H. where they convert and simplify paper-and-pencil sketch drawings to vectorized images [20].

1.3 B-Splines

Many image vectorization techniques, such as those of Hoshyari, S., et. al., and Yang, M., et. al., involves the use of fitting curves to pixels that form a border for a region in an image. The geometric primitive used in such approaches is the bézier curve [13][28][14]. A collection of bézier curves, called a b-spline, is also used for image vectorization. B-Spline is short for *basis spline*. B-splines allow for curves with higher complexity, such as those with curves resembling squiggly lines.

A k-degree B-spline curve is defined to be

$$C(u) = \sum_{j=0}^{n} c_j N_j^k(u)$$

where c_j are the control points, u is the non-decreasing knot vector $u = (u_0, \ldots, u_n)$ with u_0 and u_n having a multiplicity of $k+1$, and a B-spline functions $N_j^k(u)$. The B-spline functions are referred to as the basis functions, from which the name of b-splines was obtained from. The basis functions are defined using the Cox-de Boor recursion formula [18] which is defined to be:

$$N_j^k = \begin{cases} 1 & \text{, if } u_j \leq u \leq u_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_j^k = \frac{u - u_j}{u_{j+k}} N_{j-1}^k(u) + \frac{u_{j+k+1} - u}{u_{j+k+1} - u_{j+1}} N_{j+1}^{k-1}(u)$$

Knots in B-splines determine the basis functions, which affects the shape of the B-spline curve [18]. The number of knots in u, disregarding the multiplicity of u_0 and u_n, is defined to be

$$|u| = k + n + 1$$

B-Spline Approximation

Suppose that we are given a point sequence $p = (p_0, \ldots, p_m)$ with each point being $p_i = (x_i, y_i)$. According to the work by Laube, P., et. al. [16], we can compute the control points c_j of the B-spline curve C that will approximate p_i by using the least square problem defined as

$$\sum_{i=0}^{m} |p_i - C(t_i)|^2 \to \min$$

with precomputed parameters $t_i, i = 0, \ldots, m$ combined in the parameter vector vector $t = (t_0, \ldots, t_m)$ and end points $C(t_0) = c_0 = p_0$ and $C(t_m) = c_n = p_m$. This will give us a normal equation

$$(N^T N)c = q \tag{1.3.1}$$

where N is an $(m-1) \times (n-1)$ matrix

$$N = \begin{pmatrix} N_1^k(t_1) & \ldots & N_{n-1}^k(t_1) \\ \vdots & \ddots & \vdots \\ N_1^k(t_{m-1}) & \ldots & N_{n-1}^k(t_{m-1}) \end{pmatrix}$$

and c and q are vectors defined to be

$$c = \begin{pmatrix} c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}, \; q = \begin{pmatrix} \sum_{i=1}^{m-1} N_1^k(t_i) q_i \\ \vdots \\ \sum_{i=1}^{m-1} N_{n-1}^k(t_i) q_i \end{pmatrix}$$

and

$$q_i = p_i - N_0^k(t_i) p_0 - N_n^k(t_i) p_m$$

for $i = 1, \ldots, m-1$. If there are no constraints for end point interpolation, then 1.3.1 reduces to

$$(N^T N)c = N^T p \tag{1.3.2}$$

The control points c_j can be computed using 1.3.1 if

$$\sum_{l=1}^{m-1} N_i^k(t_i) N_j^k(t_i) \neq 0 \tag{1.3.3}$$

Chapter 2

Review of Related Literature

2.1 Works Based on Great Deluge

Many previous works dealing with course timetabling utilize an optimizing heuristic (or derivatives of it) called **Great Deluge**. Great Deluge was introduced by Gunter Dueck, and is a heuristic that is similar to Hill Climbing and Simulated Annealing. To understand how it works, imagine that you are in a point in some area with mountainous terrain. This area constitutes your solution space, with higher points in the area having higher values and, thus, having a better solution. The initial point you are located in in the area represents the initial solution that is generated. Imagine as well that it is raining endlessly and the water level W is continuously rising at a constant rate R_w, where R_w. W can start at any value that is greater than 0. Assuming that we are attempting to maximize some function Q, which evaluates a solution based on some criteria, your goal is to locate the relatively highest point in the location. This point can be thought of as the local optimum. Locating the highest point involves walking around the area that is not below the current water level. This will force you to walk to a higher and higher point since W is constantly rising. Once you are no longer able to proceed to a higher level, it means that you are now in a local optimum. Going outside the analogy and back to a technical perspective, this local optimum would now be the *relatively* best solution for your problem. Every "walk" or move to a higher point is nuanced relative to the analogy. A single walk means construction of a new solution S_{new} with basis on the current solution $S_{current}$. If $Q(S_{new}) \geq W$ [9], then we accept S_{new} as the new current solution and "walk" towards it, and we increase W by R_w. Otherwise, we simply generate a new S_{new}. These walks are performed until $Q(S_{new})$ is not greater than $Q(S_{curr})$ for a long time or we have reached the maximum number of moves/iterations [11]. Great Deluge can also be adapted to minimize Q. Instead of W increasing, it will be decreasing by the same rate. A solution S will now be accepted if $Q(S) \leq W$. Conversely, the algorithm will stop when $Q(S_{new})$ is not lesser than $Q(S_{curr})$. The second stopping condition for the algorithm still applies in this case [9][15][17]. This minimization case is the one adapted by Great Deluge-based works that focus on course timetabling.

One of the earliest works that uses Great Deluge was that of Burke, E., Bykov, Y., Newall, J., and Petrovic, S. [9]. Their work explores the use of Great Deluge in university course timetabling. They, however, slightly modified Great Deluge. Instead of simply obtaining a single new solution S in an iteration and comparing $Q(S)$ to W. Burke, E., et. al. extended Great Deluge to take multiple neighbouring solutions N on every iteration. A random solution S_N from N will be taken and it will be the one to be compared to W and the current solution. S_N will be accepted as the current solution if $Q(S_N) \leq Q(S_{curr})$. S_N can still be accepted only under the condition that $Q(S_N) \leq W$. In addition to taking multiple solutions in an iteration, a few more extensions have been added to Great Deluge by Burke, et. al.. The initial value for W is set to $Q(S_i)$, where S_i is the initial solution. Computing for R_w is made easier by using the following equation:

$$R_w = \frac{W - Q(S^{'})}{N_{moves}}$$

Chapter 3

Statement of the Problem

Raster images are composed of a pixel matrix. This makes their data representation simple. However, this reduces the amount of detail and quality they have. This limitation is clearer when scaling raster images. This motivates the use of an alternative form of representing images. One form is vector images, which uses mathematical equations to represent an image. The process of converting a raster image to a vector image is called *vectorization*. Semi-structured imagery, such as those used in graphic designs, is one of the classes of images that would benefit from vectorization. This process will allow such images to be easily scaled without sacrificing quality nor detail.

There have been numerous works that tackles image vectorization for semi-structured images. Many of these works primarily use a curve optimization algorithm such as variants of NEWUOA and conjugate gradient. A machine learning approach has been used in one of the works, but as a preprocessing step only [13]. Deep learning have been used to solve problems in multiple domains such as natural language processing (NLP), object detection, and playing board games. No other known work has applied deep learning as a core step in image vectorization. This study will deal with such application.

Chapter 4

Objectives

This study is primarily aims to apply deep learning via artificial neural networks to the problem of vectorizing semi-structured imagery. However, there are still some key objectives that this study seeks to accomplish:

1. To develop an approach that considers Gestalt psychology.
2. To evaluate the effectiveness of using deep learning for image vectorization.
3. To evaluate the accuracy of results obtained from the proposed approach to the target raster inputs.
4. To evaluate and compare the results of the proposed approach to that of previous semi-structured image vectorization methods.
5. To evaluate and compare the speed of the proposed approach compared to previous semi-structured image vectorization methods.

Chapter 5

Proposed Methodology

The proposed methodology will be based on the frameworks proposed by Hoshyari, S., et. al. [13], Yang, M., et. al. [28], Xiong, X., et. al. [25], and Laube, P., et. al. [16]. Additionally, human perception will also be taken account. Thus, the Gestalt psychology principles of accuracy, simplicity, continuity, and closure, as taken from Hoshyari, S., et. al., will be taken into account as well.

5.1 Image Preprocessing

Before a raster input image can be fitted with curves, it must preprocessed to simplify the vectorization procedure and align it with .

Corner Detection

The first step in the approach is detecting the corners in the input image. This is an important step as it will allow us to enforce the simplicity principle in Gestalt psychology and make sure the resulting vectorization be C^0 continuous should the raster input be as such.

This stage will be based from the corner detection classifier of the work by Hoshyari, S., et. al. [13]. A random forest classifier is used in the said work. Supervised learning is used as corners are manually annotated. Annotated corners will not be specific pixels. Rather, corners will be between at least two pixels. Training data is available publicly provided by the researchers. However, such data is limited only to quantized data (i.e. aliased data). The target raster input is expected to be anti-aliased data. As such, the training data will have to be built from scratch to support anti-aliased data.

Region Segmentation

In line again with the simplicity principle, the input image must be divided into regions. This will result in simpler curves being used in the final vectorization output. The additional benefit of segmenting the input into multiple distinct regions is the possibility for parallelism to be used in the vectorization approach. Since each region is distinct and independent from one another, multiple regions can be vectorized at the same time. Thus, speeding up the vectorization process.

Due to the nature of semi-structured imagery where each region will only contain a single colour, a scanline-based approach can be used in this stage. The scanline algorithm will be based off of the scanline algorithm used for boundary pixel detection in the work of Xiong, X., et. al..

For each line l_i, where $0 \leq i < h \mid h$ is the height of input image, in the raster input, each pixel p_i will be assigned to a pixel set r_{ij}, where j is the index to a set in l_i. Each pixel set will contain horizontally adjacent pixels that have the same or near-same colours. We include pixels whose colours are within a certain threshold k from the colour of the pixels that is most prominent in the set. This threshold is necessary due to the fact that certain parts of a regions may contain an anti-aliased pixel. Each l_i will have a pixel set vector r_i containing all pixel sets of l_i:

$$r_i = (r_{i0}, r_{i1}, \dots, r_{i(j-1)}, r_{ij})$$

Consequently, a region vector r will contain all pixel set vectors.

$$r = (r_0, \dots, r_i)$$

Note that there is an opportunity to utilize parallelism, as shown in the work of Xiong, X., et. al. [25], during this step due to the independent nature of every line in the raster input.

Once we obtain all the pixel sets for each line, we iterate through r, $(h - 2)$ times. For each iteration, we process r_i and r_{i+1}. If there are any r_{ij_α} whose pixels are vertically adjacent and have the same colour (or within k) to another pixel set $r_{(i+1)j_\beta}$, then those two pixel sets are merged into one. By the end of the iterations, we have obtained a set of regions that we can individually vectorize.

5.2 Curve Approximation

The core step of the proposed approach is curve approximation. This stage fits curves to the region boundaries of the raster input. This stage is based on the work by Laube, P., et. al. on curve approximation on point sequences using deep learning [16].

Point Sequence Generation

The work of Laube, P., et. al. takes a point sequence as input. As such, for this proposed approach, we must generate point sequences from the region we will be vectorizing. For every region we ought to vectorize, we treat the center of boundary pixels and the detected corners of each region as a point sequence. However, we must also take into account the fact that certain portions of a region may be a corner where the curves in such segment would have C_0 continuity. As such, the point sequence we generate must take into account corners. This would implore us to take note of the following cases during point sequence generation:

1. For regions with no corners, a random pixel will be selected as both start and end point of the point sequence. The expectation is that there will be no difference in the resulting curve from choosing a different start and end point.

2. For regions with a single corner, the corner point will be selected as the start and end point of the point sequence. This is to ensure that the resulting vector output will have a corner at that point.

3. For regions with two or more corners and assuming n is the number of corners, the point sequence will be divided at those corners into separate point sequences. This will result in $(n + 1)$ new point sequences. Each new point sequence will have their start and end points be the corner points they are adjacent to.

Each point sequence will then be passed to the next stage to be parametrized and have a curve approximated for.

Point Sequence Curve Approximation

The point sequences obtained from the previous step are now to be fitted with curves, specifically B-splines. As provided by the framework by Laube, P., et. al., two neural networks will be used in this stage and some preprocessing will be performed on the point sequences. The two neural networks are a point parametrization network (PPN), which approximates parametic values to point sequences, and a knot selection network (KSN), which predicts new knot values for knot vector refinement.

Sequence Segmentation

The input point sequence must first be split. This is to ensure that real data and training data match in terms of complexity. Let us define a function $\hat{k}(p)$ that measures the complexity of a point sequence p, where k_i is the curvature at point p_i, given its total curvature:

$$\hat{k}(p) = \sum_{i=0}^{m-1} \frac{(|k_i| + |k_{i+1}|)||p_{i+1} - p_i||_2}{2}$$

A point sequence p is split into point sequence segments $p^s, s = 1, \ldots, r$ at the median, if $k\hat{(p)} > \hat{k_t}$ for a threshold $\hat{k_t}$. This $\hat{k_t}$ will be set, as per the original authors have done, to the 98th percentile of $\hat{k}(\cdot)$ of the training set. This process is performed $r - 1$ times, until each p^s satisfies $k\hat{(p)} > \hat{k_t}$.

Sub/Supersampling and Normalization

To be able to approximate parametric and knot values using the PPN and KSN, the number of points per segment p^s must equal the input size l of the aforementioned networks. As such, all segments p^s are either subsampled or supersampled.

If a segment p^s has a number of points greater than l, then p^s is subsampled. This process involves drawing points in p^s such that the drawn indices i are equally distributed and include the first and last point. If, on the other hand, the number of points in p^s is less than l, then *temporary* points are linearly interpolated between consecutive points p_i^s and p_{i+1}^s. This interpolation is performed until the number of points equal l.

The sampled segments are then normalized to $\bar{p}^s$, which consists of the points

$$\bar{p}_i^s = \frac{p_i^s - min(p^s)}{max(p^s) - min(p^s)}$$

where $min(p^s)$ and $max(p^s)$ are the minimum and maximum coordinates of p^s respectively.

Parametrization of Point Segments

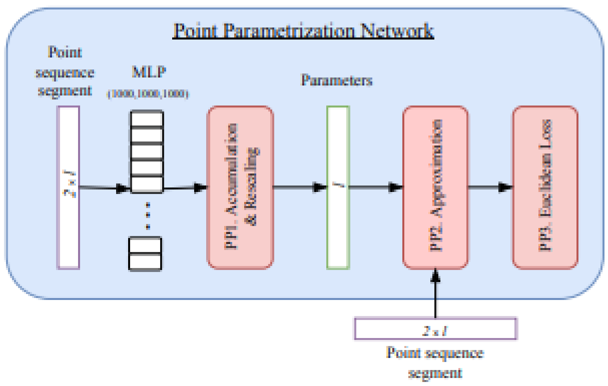

Figure 5.1: The network architecture of the Point Parametrization Network (PPN)

.

The PPN will be responsible for parametrization of point segments. For every $\bar{p}^s$, the PPN generates a parametrization $\bar{t}^s \subset [0, 1]$, which will be rescaled to $[u_{s-1}, u_s]$ and adapted to sampling of $\bar{p}^s$.

In supersampled segments, the parameters t_i^s of temporary points are simply removed from $\bar{t}^s$. In a subsampled p^s, for every point p_i that was removed from the segment, a parameter $\bar{t}_i$ is inserted to $\bar{t}^s$.

$$t_i = t_\alpha^s + (t_\beta^s)\frac{chordlen(p_\alpha^s, p_i)}{chordlen(p_\alpha^s, p_\beta^s)}$$

chordlen is the length of the polygon defined by a point sequence. In the subsampled segment, with parameters t^s_α and t^s_β, p^s_α and p^s_β are the closest neighbours of p_i.

The initialization of the parametric step requires an initial knot vector. We first define $u_0 = 0$ and $u_n = 1$. For each segment (except the last one), one knot u_i is added.

$$u_i = u_{i-1)} + \frac{chordlen(p^s)}{chordlen(p)}, i = 1, \ldots, r-1$$

This yields a start and end knot for every point sequence segment.

The PPN Architecture The PPN, as stated earlier, takes in an input of segments p, which can be written as $p = (x_0, \ldots, x_{l-1}, y_0, \ldots, y_{l-1})$. The parameter domain is defined as $u_0 = t_0 = 0andu_n = t_{l-1} = 1$. For a sequence of points p, a parameter vector $t = (t_i)_i$, is defined as $t_i = t_{i-1} + \Delta_{i-1}$. The task of the PPN is to predict missing values $\Delta = (\Delta_0, \ldots, \Delta_{l-2})$ with

$$\Delta_s ubi > 0, i = 0, \ldots, l-2$$

such that $t_0 < t_1$ and $t_{l-2} < t_{l-1}$. We apply a multilayer perceptron (MLP) to the input data p, yielding as output a distribution for parametrization $\Delta^{mlp} = (\Delta^{mlp}_0, \ldots, \Delta^{mlp}_{l-2})$ of size $l-1$.

The PPN further contains additional layers PP1, PP2, and PP3, which will be discussed next.

PP1. Accumulation and Rescaling The output Δ^{mlp} is used to compute a parameter vector t^{mlp} with $t^{mlp}_0 = 0$ and

$$t_i^{mlp} = \sum_{j=0}^{i-1} \Delta_j^{mlp}, i = 1, \ldots, l-1$$

Since t_{l-1}^{mlp} is usually not 1, rescaling t^{mlp} yields the final parameter vector t with

$$t_i = \frac{t_i^{mlp}}{max(t^{mlp})}$$

The MLP layer in the PPN uses a softplus activation function defined to be:

$$f(x) = ln(1 + e^x)$$

PP2. Approximation B-spline curve approximation is included directly into the PPN as a network layer. The input points p and their parameters t are used for an approximation with knot vector $u = (0, 0, 0, 0, 1, 1, 1, 1)$ for $k = 3$. The approximation layer's output $p^{app} = (p_0^{app}, \ldots, p_{l-1}^{app})$ is the approximating B-spline curve evaluated at t.

PP3. Euclidean Loss A loss function, which is a Euclidean loss function, is to be used in the PPN. The Euclidean Loss function is defined as

$$\frac{1}{l} \sum_{i=0}^{l-1} ||p_i - p_i^{app}||_2 \tag{5.2.1}$$

Parametrization Refinement

In some cases, the approximated parametrization of a p^s may have errors. The approximation error of a p^s is computed by using the Hausdorff distance to the input data p.

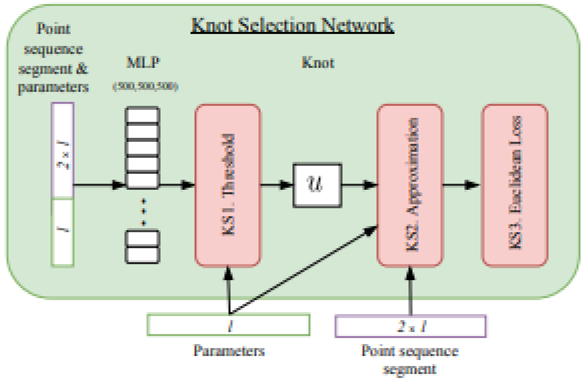

Figure 5.2: The network architecture of the Knot Selection Network (PPN)

.

The segments p^s that have a large approximation error are computed a new knot using the KSN. For $\bar{p}^s$ and $\bar{t}^s$, the KSN generates a new estimated knot $\bar{u}_s \in [0, 1]$. The new knot $\bar{u}_s$ is mapped to the actual knot value range $[u_{s-1}, u_s]$ by

$$\tilde{u}_s = u_{s-1} + \bar{u}_s(u_s - u_{s-1})$$

Instead of $\tilde{u}_s$, the parameter value t_i closest to $\tilde{u}_s$ is inserted into u. u can be further refined until the desired curve approximation error threshold is satisfied.

The KSN Architecture This stage utilizes a KSN, as mentioned earlier. The KSN predicts a new knot u to the interval $(0, 1)$ for a given segment p and parameters t, of which were previously approximated by the PPN. This network uses an MLP, which transforms the input to a single output value u^{mlp}. The RELU function is used as the activation function for the MLP, except for the output layer where the Sigmoid function is used instead.

Similar to that of the PPN, the KSN has three additional layers: KS1, KS2, and KS3.

KS1. Threshold Layer The new knot u computed has to satisfy the following conditions: $u \in (0,1)$, $t \cap [0,u] \neq \theta$, and $t \cap [u,1] \neq \theta$. Satisfying these conditions will will require us to use a threshold layer which maps u^{mlp} to

$$u = \begin{cases} \epsilon & \text{, if } u^{mlp} \leq 0 \\ 1-\epsilon & \text{, if } u^{mlp} \geq 1 \\ u^{mlp} & \text{, otherwise} \end{cases}$$

Introducing a small $\epsilon = 1e-5$ makes sure that the knot multiplicity at the end knots stays equal to k.

KS2. Approximation Approximation in the KSN is generally similar to that of PPN. The only different is that of the knot vector. The knot vector in the KSN is defined to be $u = (0,0,0,0,u,1,1,1,1)$. For backpropagation, the derivative of the B-Spline basis functions with respect to u is required.

KS3. Euclidean Loss The loss function for the KSN is the same as that of the PPN. See 5.2.1.

Network Training

The training of the PPN and KSN will be based from the work of Laube, P., et. al. [16]. The input size of the network will be $l = 100$.

The data set generated by the original authors consists of 150,000 curves. This data was synthesized from B-spline curves. Random control points c_i were generated using a normal distribution μ and variance δ to define cubic ($k = 3$) B-spline curves with $(k+1)$-fold end knots and no interior knots. The y-coordinates are given the configuration: $\delta = 2$ and $\mu = 10$. For the x-coordinates, $\delta = 1$ and $\mu = 10$ are used for the first control point. All consecutive points have μ increased by $\Delta\mu = 1$. Curves with self-intersections are discarded, because the sequential order of their sampled points is not unique, and point sequences are usually split into subsets at the self-intersections. Smaller δ for the x-coordinates of control points reduces the number of curves with self-intersections. To closely match the target input as much as possible, we also include curves that have been manually fitted to raster images. These curves can be obtained from vector images available online.

For each curve, l points $p = (p_0, \ldots, p_{l-1})$ are sampled. These curves then to have increasing x-coordinates from left to right. As such, index-flipped versions of the point sequences of the dataset are added, resulting in 300,000 point sequences. 20% of the sequences are used as test data in the training process.

The PPN is trained first since the KSN requires point parametrizations t, which is obtained from the PPN. After training, the PP2 and PP3 layers are discarded and PP1 becomes the output layer of the PPN. The parametric values t are computed for the training dataset by applying the PPN and train the KSN on the combined input. After training, KS2 and KS3 are discarded, with KS1 becoming the network output layer. The MLPs of the PPN and KSN will consist of three hidden layers with sizes (1000, 1000, 1000) and (500, 500, 500) respectively. Dropout is applied to the MLP layers. The network is trained using the Adam optimizer.

5.3 Region Colouring and Merging

Once the curves have been approximated, the vectorization of the region will be filled with the colour prominent in the raster version of the region. The regions will be plotted unto their locations in the original raster input. Once all the regions have been plotted, they will be grouped into a single vectorization. This will now be the vectorization output of the raster image.

Chapter 6

Describing How You Validated Your Approach.

Chapter 7

Stating Your Results and Drawing Insights From Them.

Chapter 8

Summarizing Your Thesis and Drawing Your Conclusions.

Appendix A

What should be in the Appendix

What goes in the appendices? Any material which impedes the smooth development of your presentation, but which is important to justify the results of a thesis. Generally it is material that is of too nitty-gritty a level of detail for inclusion in the main body of the thesis, but which should be available for perusal by the examiners to convince them sufficiently. Examples include program listings, immense tables of data, lengthy mathematical proofs or derivations, etc.

Bibliography

[1] 6.22.scale image. https://docs.gimp.org/en/gimp-image-scale.html.

[2] Photoshop image size and resolution. https://helpx.adobe.com/photoshop/using/image-size-resolution.html{\\#}resampling.

[3] Titan xp graphics card with pascal architecture — nvidia geforce. https://www.nvidia.com/en-us/titan/titan-xp/.

[4] Raster vs vector. https://vector-conversions.com/vectorizing/raster_vs_vector.html, n.d.

[5] Raster vs. vector graphics: What's the difference? https://pixellogo.com/blogs/pixellogo-blog/raster-vs-vector-graphics, n.d.

[6] P. J. Barendrecht, M. Luinstra, J. Hogervorst, and J. Kosinka. Locally refinable gradient meshes supporting branching and sharp colour transitions. *The Visual Computer*, pages 949–960, 2018.

[7] M. Bessmeltsev and J. Solomon. Vectorization of line drawings via polyvector fields. *CoRR*, abs/1801.01922, 2018.

[8] T. Birdal and E. Bala. A novel method for vectorization. *CoRR*, abs/1403.0728, 2014.

[9] Edmund Burke, Yuri Bykov, James Newall, and Sanja Petrovic. A time-predefined approach to course timetabling. *Yugosl. j. oper. res.*, 13:139–151, 01 2003.

[10] Computerphile. Resizing images. `https://www.youtube.com/watch?v=AqscP7rc8{\\_}M`, 2016.

[11] Gunter Dueck. New optimization heuristics: The great deluge algorith and the record-to-record travel. 104:86–92, 01 1993.

[12] Juan Carlos Gonzales. Use of convolutional neural networks for image classification. `https://www.apsl.net/blog/2017/11/20/use-convolutional-neural-network-image-classification/`, 2017.

[13] S. Hoshyari, E. A. Dominici, A. Sheffer, N. Carr, Z. Wang, D. Ceylan, and I. Shen. Perception-driven semi-structured boundary vectorization. *ACM Transactions on Graphics*, 37(4):1–14, 2018.

[14] J. Kopf and D. Lischinski. Depixelizing pixel art. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011)*, 30(4):99:1 – 99:8, 2011.

[15] Dario Landa-Silva and Joe Henry Obit. Great deluge with non-linear decay rate for solving course timetabling problems. 01 2008.

[16] P. Laube, M. Franz, and G. Umlauf. Deep learning parametrization for b-spline curve approximation. *CoRR*, abs/1807.08304, July 2018.

[17] Joe Henry Obit, Dario Landa-SIlva, Marc Sevaux, and Djamila Ouelhadj. Non-linear great deluge with reinforcement learning for university course timetabling. 01 2011.

[18] Mathematics of Computer Graphics and Virtual Environments. B-splines, 2015.

[19] P. Parsania and P. Virparia. *International Journal of Innovative Research in Computer and Communication Engineering*, (12):7409.

[20] E. Simo-Serra, S. Iizuka, K. Sasaki, and H. Ishikawa. Learning to simplify: Fully convolutional networks for rough sketch cleanup. *ACM Trans. Graph.*, 35(4):121:1–121:11, July 2016.

[21] J. Sun, L. Liang, F. Wen, and H. Shum. Image vectorization using optimized gradient meshes. Association for Computing Machinery, Inc., July 2007.

[22] Techquickie. How do vector graphics work? `https://www.youtube.com/watch?v=W2xknX3k6FY`, 2016.

[23] Y. Wang, F. Perazzi, B. McWilliams, A. Sorkine-Hornung, O. Sorkine-Hornung, and C. Schroers. A fully progressive approach to single-image super-resolution. June 2018.

[24] G. Xie, X. Sun, X. Tong, and D. Nowrouzezahrai. Hierarchical diffusion curves for accurate automatic image vectorization. *ACM Trans. Graph.*, 33(6):230:1–230:11, 2014.

[25] X. Xiong, J. Feng, and B. Zhou. Real-time image vectorization on gpu. *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2016)*, 1:143–150, 2016.

[26] A. Yang. A.i. gigapixel an inside story topaz labs. `https://topazlabs.com/a-i-gigapixel-story/`, 2018.

[27] C. Yang, C. Ma, and M. Yang. Single-image super-resolution: A benchmark. pages 372–386. Springer International Publishing, 2014.

[28] M. Yang, H. Chao, C. Zhang, J. Guo, L. Yuan, and J. Sun. Effective clipart image vectorization through direct optimization of bezigons. *IEEE Transactions on Visualization and Computer Graphic*, Feb. 2016.