Single-Class Instance Segmentation for Vectorization of Line Drawings

by

Rhythm Vohra

A Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

University of Victoria

Single-Class Instance Segmentation for Vectorization of Line Drawings

by

Rhythm Vohra

B.E., Panjab University, 2014

Supervisory Committee

Dr. Alexandra Branzan Albu, Supervisor

(Department of Electrical and Computer Engineering)

Dr. Pan Agathoklis, Departmental Member

(Department of Electrical and Computer Engineering)

ABSTRACT

Images can be represented and stored either in raster or in vector formats. Raster images are the most ubiquitous and are defined as matrices of pixel intensities/colours, while vector images consist of a finite set of geometric primitives, such as lines, curves, and polygons. Since geometric shapes are expressed via mathematical equations and defined by a limited number of control points, they can be manipulated in a much easier way than by directly working with pixels; hence, the vector format is much preferred to raster for image editing and understanding purposes. The conversion of a raster image into its vector correspondent is a non-trivial process, called image vectorization. Creating vector images from a given raster image can be time-consuming and requires the expertise of a skilled graphic user.

This thesis explores the effectiveness of a Deep Learning (DL) based framework to vectorize raster images comprising line drawings with minimal user interventions. To improve the visual representation of the image, each stroke in the line drawing is represented with a different label and vectorized. In this document, we present an in-depth study of image vectorization, the objective of our research, challenges, potential resolutions, and compare the outcomes of our approach on six datasets consisting of different types of hand drawings. More specifically, this thesis begins by comparing raster images with vector images, the importance of image vectorization, and our objective to convert raster images to vector-based representations by accurately separating each stroke from the line drawings.

In further chapters of this thesis, a Deep Learning (DL) based segmentation methodology is introduced to perform Single-Class Instance Segmentation (SCIS) of hand drawings to process the input raster image by labeling each pixel as belonging to a particular stroke instance. This segmentation approach is able to leverage the spatial relationships between each stroke instance.

A novel loss function specifically designed to optimize our highly imbalanced datasets by scaling the margins and adding a regularization term to improve its feature selection

technique. The weighted combination of our proposed margin regularized loss function is combined with the Dice loss to reduce the spatial overlap and improve the predictions over infrequent labels.

Finally, the effectiveness of our segmentation technique of line drawing vectorization is compared experimentally with the state-of-the-art and our reference method. Our method can successfully handle a wide variety of human drawing styles. The results are comparable in terms of accuracy and way ahead in terms of speed and complexity, with other methods.

Contents

List of Tables

List of Figures

Acronyms

AG Attention Gates. xii, 25, 28, 31, 33

CAD Computer-Aided Design. 13, 73

DL Deep Learning. iii, vii, 58, 59

FCN Fully Convolutional Networks. x, 23–25

FN False Negative. xiv, 19, 45

FP False Positive. xiv, 19, 45, 46

HT Hough Transform. 12, 13

IoU Intersection over Union. vii, xv–xx, 19, 20, 54, 56–59, 66–70

LDAM Label-Distribution-Aware Margin. 20, 41, 43

LSTM long short-term memory. 35

MFAG Multi-Focus Attention Gates. vi, xii, xiii, 9, 22, 28, 30–32, 34, 36, 71

MFAU Multi-Focus Attention UNet. xii, 9, 22, 27, 28, 31, 47, 71

MRF Markov Random Field. 17

MRLF Margin-Regularised Loss Function. vii, 39, 58, 59

OZZ Orthogonal Zig-Zag. 13

ReLU Rectified Linear Unit. 16, 33

RHT Randomized Hough Transform. 12

SCIS Single-Class Instance Segmentation. iii, 6–8, 17, 22, 29, 42, 71

TP True Positive. xiv, 45

ACKNOWLEDGEMENTS

It has been a journey of transformation to finish this thesis, and I'd like to thank all those who have supported me along the way. First of all, I'd like to express my deepest gratitude to my supervisor, Dr. Alexandra Branzan Albu. I appreciate the expertise, patience, and commitment you've shown me to my growth as a researcher. Your mentorship has not only shaped this thesis but also my academic and professional development. It's really a blessing for me to have you as my supervisor.

A special thanks to Amanda Dash, for all her guidance and advice throughout this research. This work wouldn't have been possible without her. Also, I would like to thank Melissa Cote for all her unconditional support throughout the thesis.

I would also like to thank all my lab members at the Computer Vision lab, at the University of Victoria for making this journey memorable.

This thesis is dedicated to:

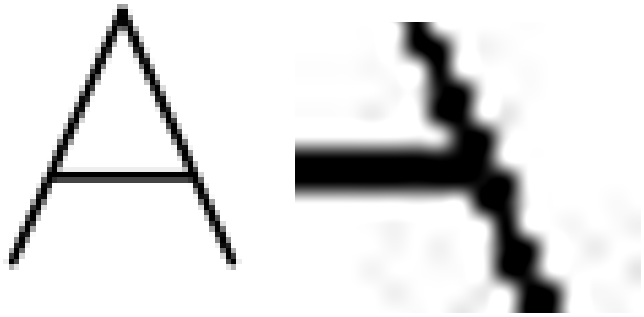
my Mom and Dad ♥

Chapter 1

Introduction

1.1 Overview

Images are rich sources of information and can be stored in either raster or vector formats. Raster images are made up of individual pixels arranged in a grid or an array. Each individual pixel consists of a unique color or tonal information, and when combined, they form a complete image. Cameras use sensors that capture the real-world scene in a grid of photosensitive elements (with each of these elements often referred to as pixels in the image), and digital screens are engineered to display images as an array of pixels. As a result, most of the images available to us are stored in raster format. Raster images have applications in various contexts, including digital photography, print, media, web designs, screens, and many others. Since raster images consist of a fixed number of pixels, when zoomed-in or modified, the quality of the raster image may degrade (as shown in Figure 1.1). The standard file formats used to store raster images include .tiff, .jpg, .png, and .bmp. A high-quality raster image may consist of thousands to millions of pixels; thus raster images are difficult to edit at a pixel level. Additionally, these images require ample storage space. Suppose we have a higher-resolution image with millions of pixels, with each pixel



(a) Raster Image (b) Zoomed-in Raster Image

(c) Vector Image (d) Zoomed-in Vector Image

Figure 1.1: Comparison between vector and raster images when zoomed in; vector images maintain clean lines, while raster images get blurred because of constraints in resolution. (a) Raster image, (b) zoomed-in raster image, (c) vector image, (d) zoomed-in vector image.

taking up to 32 bits of space; the total storage space required for an image will be 1000 X 1000 X 32 (Total number of pixels X Bits per pixel). To save storage space, images can be compressed using various compression algorithms. These compression algorithms are based on the assumption about how pixels vary within an image. Instead of storing pixels individually, the algorithms take advantage of their similarities by encoding them using a more efficient representation. For example, green grass in an image will have many similar color pixels; a compression algorithm may try to group these pixels together and represent them as a single value, creating a compact image. However, a trade-off exists between a high compression ratio and a high-quality image. To overcome these trade-offs, vector images were introduced. They use less storage space while maintaining high image quality.

Vector images are generated using mathematical formulas rather than pixels, as in the

case of raster images. These images are described by geometric primitives such as curves and their control points. Bezier curves are used most frequently; they are based on Bernstein polynomials [4], which are used to interpolate between control points and were introduced by Paul de Casteljau [4] to construct smooth and continuous shapes. Figure 1.2 illustrates different types of Bezier curves. These curves are defined by a set of control points $P \in 1, 2, ..., n$, where n-1 is the order of the curve (for n = 2, the order of the curve is linear, n = 3, the curve is quadratic, n = 4, it is a cubic curve and so on). The first and last control points define a curve's starting and ending coordinates. While the rest of the control points do not lie on the curve, they are used to influence the direction and shape of the curve. Tangent t traverses through the curve segment by varying from 0 to 1, used to interpolate between the control points to generate a smooth curve. The mathematical equation below is used to evaluate the Bezier curve:

$$B(t) = \sum_{0}^{n} b_{i,n}(t) P_i \tag{1.1}$$

where, $t \in [0, 1]$, P_i are Bezier control points and $b_{i,n}$ are the Bernstein polynomials of degree n, defined as:

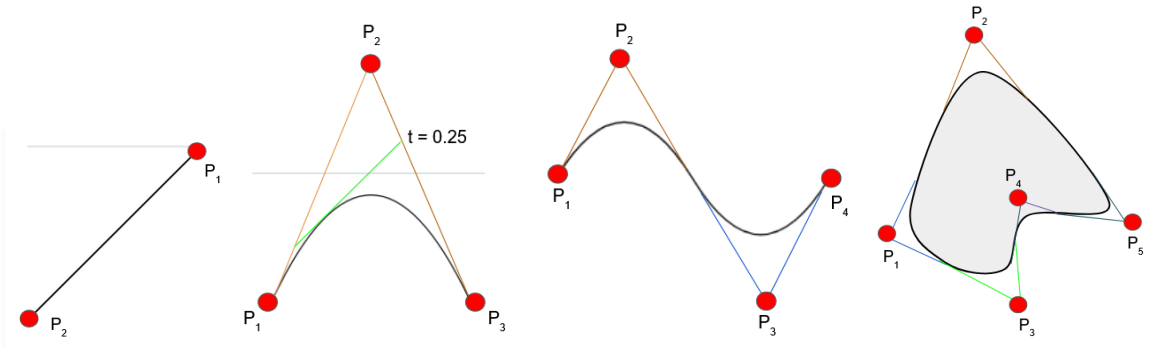
$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \qquad i = 1, 2, ..., n \tag{1.2}$$

For the linear Bezier curve with degree n as 2, the equation is:

$$B(t) = (1-t)P_1 + tP_2, \qquad 0 \leq t \leq 1 \tag{1.3}$$

The equation for the quadratic Bezier curve (n = 3) is:

$$B(t) = P_2 + (1-t)^2(P_1 - P_2) + t^2(P_3 - P_2), \qquad 0 \leq t \leq 1 \tag{1.4}$$



(a) Linear Curve (b) Quadratic Curve (c) Cubic Curve (d) Higher-order Curve

Figure 1.2: Various types of Bezier curve in vector graphics. (a) Two control points are used as extremities in a linear curve, (b) three points for a quadratic curve, and t is the tangent of the curve segment that varies from 0 to 1 by interpolating between control points to generate a smooth shape, (c) four control points are used to change the shape of the cubic curve, and (d) high-order curve represented by 5 number of control points that can be used to influence the shape of the Bezier curve, also referred to as paths.

Vector images have phenomenal flexibility because they can be stretched, rotated, or scaled while not compromising their clarity, just by changing the position of control points. Editing these mathematically described shapes is much simpler than editing pixels in a raster image. The edited images maintain their sharpness across a wide range of resolutions, a highly desirable attribute (see Figure 1.1). Since it is easier to store mathematically described shapes, vector images require far less storage space than storing thousands to millions of pixels in a raster image. Figure 1.3 illustrates a comparison between a shape stored in raster and vector formats. As can be seen from the figure, representing this simple shape in raster format requires 12x8 pixels, while the vector image was constructed by using only 4 points. .ai, .svg, .pdf, .eps, and many others are among the most commonly used formats for vector images.

Despite several advantages of vector images over raster images, many graphical users prefer drawing by hand using a pen or stylus because it feels more natural and gives them control over their work. As a result, there is a significant need for converting hand-drawn raster images into vector formats. Commercial vectorization tools such as InkSpace, Corel-

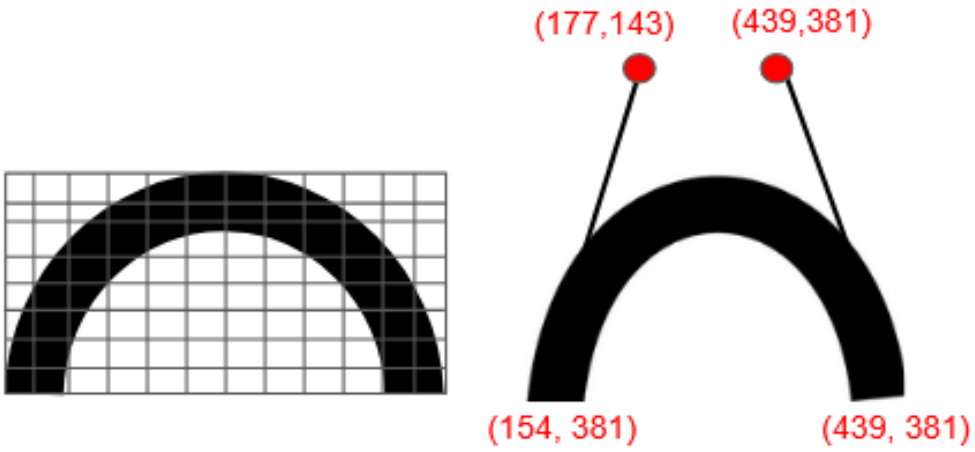


Figure 1.3: Comparison between raster and vector image with respect to storage space. *Left*: the shape requires 12x8 pixels in raster format. *Right*: the shape can be stored using just 4 control points in vector format.

DRAW, and AdobeTrace use manual tracing of the vector lines from the raster drawings by a skilled user. However, this manual tracing process can be labor-intensive and time-consuming.

There have been several attempts made by researchers to perform a non-trivial process called image vectorization (the process of converting raster images to vector images) automatically. While performing image vectorization, multiple vector configurations can produce the same raster image. For example, in Figure 1.4, it is shown that a rectangle can be made up of different possible configurations, such as two sets of parallel segments or three connecting lines and a single one, or by combining two separate bend lines or four separate segments. However, we know the most likely way to draw a rectangle is with four different lines, by understanding the typical way in which most users draw a rectangle. Despite being visually coherent, there can be possible vector configurations that can lead to a "vector soup", i.e., a collection of numerous small vector paths (example shown in Figure 1.5). These vector ambiguities can make any further edits challenging. Therefore, it is necessary to have a clear understanding of the content being drawn.

In this work, we are considering a particular type of image containing line drawings (a

Figure 1.4: Possible configurations in which strokes can be segmented to represent a rectangle with the first rectangle being the raster image and the last one being the most likely way to draw a rectangle by the user.

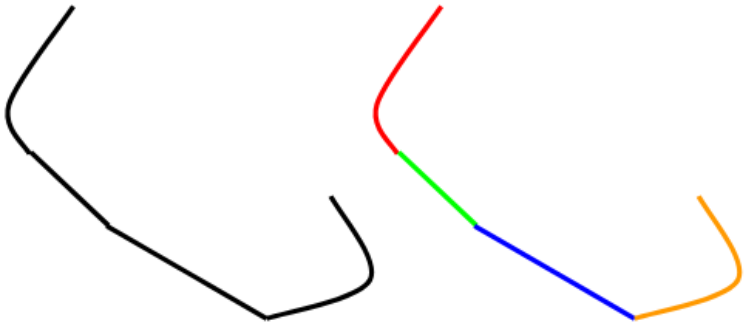


Figure 1.5: Simple example of vector soup for a curve. *Left*: ideal way the curve should be represented, i.e. as a single vector path. *Right*: multiple paths representing the same vector path, resulting in the vector image being challenging to edit.

fundamental artistic form in the field of graphical design), such as characters and sketches. These images are composed of a combination of various strokes. Strokes are a sequence of continuous sets of points created with a writing or drawing instrument (pen, stylus) as long as that instrument remains in contact with the surface. By successfully implementing stroke segmentation of line art drawings, it is possible to achieve a precise and easy-to-edit representation of the original raster image. Figure 1.6 shows an example of a raster image and its vectorized version produced by segmenting each stroke accurately and then vectorizing those strokes. In images containing characters such as Chinese and Kanji, the accurate segmentation of strokes is important, but it is also important to maintain the correct ordering of the strokes. As a result, the accurate segmentation of strokes along with preserving the sequence of strokes is a priority for us.

We aim to perform Single-Class Instance Segmentation (SCIS), that is, we have a single

Figure 1.6: Representation of raster image (left) and vector image with each instance of stroke segmented (right).

class known as foreground (i.e., strokes), and unique labels are assigned to each subsequent stroke (i.e., stroke instance). However, performing SCIS of strokes can be challenging, mainly when there is a considerable variation between the sizes and shapes of a stroke. There can be various ways in which strokes can be represented, depending upon the different writing styles of the user. Figure 1.7 shows an example of an instance segmentation on a cat sketch. There are various possible ways to segment a cat sketch. Ideally, the two ears, the connecting line between these ears, the face, and each cat whisker are segmented as a separate stroke. However, the style in which the user has drawn this cat depicts that the first ear and the face curve is one stroke, the second ear and the connecting line between the ears as second stroke, and each whisker as four different strokes. Moreover, in the case of overlapping strokes, it is challenging to assign which overlapped component belongs to which stroke during the segmentation task. Consider the example in Figure 1.8 where intersections of four paths and two paths are shown. Segmenting the sketches into each stroke instance is essential for extracting specific information from stroke-based images to produce their final vector counterparts.

This thesis provides several contributions to perform instance segmentation of strokes and then vectorize these strokes, as outlined in the following section.

Figure 1.7: Possible configurations in which strokes can be segmented to represent a cat with the first cat being the raster image and the last one is the correct implementation of the segmented cat.

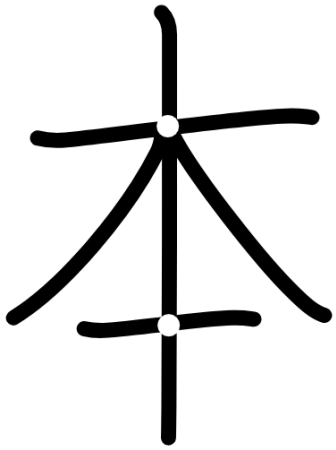


Figure 1.8: Top and bottom white dots indicate four and two overlapping paths, respectively.

1.2 Thesis Objectives and Contributions

We have seen in Section 1.1 that a raster image of a line drawing needs to be segmented accurately in order to convert it to a vector image.

The main objective of this thesis is to perform Single-Class Instance Segmentation (SCIS), i.e., classification of each stroke individually, to convert hand-drawn strokes (i.e. instances) from binary (one-class) raster images into vector-based stroke representations. We propose a segmentation method that processes the input raster image by labeling each pixel as belonging to a particular stroke instance. The labeled regions in the output of our segmentation model are separately vectorized using a commercial tool called Potrace [5].

Figure 1.9: Vectorization by segmenting each stroke instance of a line drawing. *Left to Right:* Grayscale raster image; segmented output of raster image; instances separated and vectorized, (relative position intentionally modified for visualization purposes) final vectorized result.

The vectorized forms of all strokes are then combined to form the final vector image.

Figure 1.9 shows the basic steps of our method. A grayscale raster image is labeled as it passes through our proposed segmentation model. The labels are separated and represented in a bitmap format from the resultant segmented raster image. This bitmap format is vectorized using the Potrace software [5] (discussed in Section 4.2). Each vectorized segment is combined, considering the exact positioning of strokes to produce the vectorized version of the original raster image.

The main contributions of this thesis are:

- We propose a novel segmentation model named Multi-Focus Attention UNet (MFAU) with two specific contributions that include 1) Multi-Focus Attention Gates (MFAG), and 2) Highway Skip Connections, outperforms both state-of-the-art and reference methods across several datasets using various performance metrics.

- We propose a novel loss function, the Margin-Regularised Loss Function (MRLF), that generalizes well on less frequent labels in a highly imbalanced dataset. We combine this MRLF with a spatial-based loss function (Dice loss) to create a compounded loss function. This approach enables us to generate outputs consistent with the user's drawing styles while implying the perceptual grouping principles of similarity, continuity, and closure.

Chapter 2 reviews related works on the vectorization of line drawings, and loss functions. Chapter 3 discusses our contributions, namely the segmentation model and loss function. Chapter 4 discusses and compares our results both qualitatively and quantitatively to the state-of-the-art method. Chapter 5 concludes the thesis and offers future work recommendations.

1.3 Publications

The findings of the research described in this thesis have been submitted to the ***International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*** as a 12-page manuscript in IEEE format. As the main author of the paper, titled ***"Single-Class Instance Segmentation for Vectorization of Line Drawings"***, I was responsible for the research design and experiments, as well as for writing most of the paper. My co-authors contributed with recommendations for research and experiments as well as with several iterative revisions of the manuscript.

Chapter 2

Literature Review

This chapter reviews the current progress of the vectorization of hand drawings. We have divided this chapter into two parts. Section 2.1 discusses recent works in vectorization of line art drawings. Section 2.2 reviews the works related to the loss function.

2.1 Vectorization Techniques

Vectorization, i.e. the automatic conversion of raster images into their vector counterparts, has been a popular research topic for a long time due to the advantages of the vector over raster formats as mentioned in the previous chapter. This section first discusses vectorization methods that focus on detecting geometric primitives with traditional computer vision and image processing techniques, then discusses recent deep-learning based approaches.

2.1.1 Traditional Computer Vision Approaches

Earlier works in the field of vectorization combine a variety of image processing operators and techniques such as thinning, thresholding, layering, contour finding, edge detection, curve fitting, feature point extraction, Hough transform, and polygonal approximation. In 1982, J. Jimenez *et al*. [6] conducted experiments for image vectorization using contour

following (tracing the boundary of an object), thinning (iteratively removing pixels from the contours to create a skeleton-like structure), and polygon approximation (simplifying a contour by approximating it with a polygon) that is specifically designed to handle complex images. They experimented with the Map Manipulation System, a package developed at IBM, to explore the relationship between the raster and vector images and display the vectorized results obtained using the integration of three discussed algorithms (i.e., contour following, thinning, and polygon approximation) on the display screens.

In 1990, P. Kultanen *et al.* [7] used the Randomized Hough Transform (RHT) algorithm to vectorize line drawings by accurately extracting curves from the images and then converting them into vector representations. RHT is a variant of the Hough Transform (HT), where a single point in parameter space that represents a curve in the image can be distinctly defined by using a pair, triple, or multiple points from the image space. These image points are selected through random sampling of the pixels from the input image. Depending on the application, they proposed that RHT can be used for curve extraction before/after the thinning process. These curves can then be passed through the vectorization phase (this step is not included in the paper).

In 1996, M. Roosli *et al.* [8] performed vectorization using segmentation and error fitting. During the image segmentation process, specific segmented parts of the raster image are referred to as raw primitives. To add meaningful relationships to the raw primitives, the authors use a data structure. This data structure organizes and associates parts of the raster image with each raw primitive. The structure also includes ”pointers” from the raw primitives back to the raster image. These pointers are used to assess how well a raw primitive aligns with the corresponding region in the original raster image. An error fitting process is performed on these raw primitives to generate final primitives by using least square fitting optimization. Additionally, constraints are introduced to the tangents of the raw primitives during error fitting to fit a curve for accurate vectorization.

In 1997, D. Dori *et al*. [9] proposed an Orthogonal Zig-Zag (OZZ) algorithm for vectorizing engineering drawings that focuses on line recognition and performs much better, with significantly lower computational complexity, than the HT. This algorithm detects lines in binary images by using the same principle as the propagation of a ray of light through optical fiber. The width of the ray is one pixel and it traverses through the black pixels (treating them as a conducting path), orthogonally in a zig-zag manner. Whenever the ray encounters a white pixel, its direction changes by 90°. While the ray traverses through the image, it collects information about the presence, width, and start-end points, and validates that a junction or crossing has been encountered.

In 2000, K. Tombre *et al*. [10] organized the overall process of vectorization into three mandatory steps and explored different methods to solve these processes: (1) First, extracting lines or chains of pixels from a raster image, with three main approaches: skeleton-based methods (offer good precision but can be sensitive to irregularities), line matching methods (better at positioning junction points but rely on heuristics), and sparse-pixel methods (use sub-sampling but may lead to double detections in some cases). (2) Approximating these lines into a set of vector paths. This can be achieved using different polygon approximation methods such as maximum distance criterion, algebraic area, angular measures, or curvature computations, each with its own advantages and limitations. (3) Post-processing to refine the junction points and merge or eliminate some of the vectors to improve the quality of the results. These post-processing steps may take an ad hoc approach, designed specifically for a particular problem. This can involve adding constraints to idealize the image, heuristic corrections to achieve good results with added thresholds, or performing generic and robust corrections.

In 2007, A. Bartolo *et al*. [11] proposed a technique to vectorize scribbled drawings for Computer-Aided Design (CAD) interpretation which is compatible with the natural drawing habits of the user. They used Gabor filters to simplify the scribbles, and then

extracted vector lines with a Kalman filter. A Gabor filter is a special case of a bandpass filter, designed to be selective in terms of frequency (i.e., it responds differently to various patterns) and orientation (i.e., it responds differently to lines at different angles) in the input. Based on this response, the scribbles are traced into one line and therefore, used to simplify scribble drawing. In order to perform vectorization of the simplified lines, the Kalman filter is applied for line tracking. This filter makes recursive estimations of the information from the Gabor filter and combines the multiple measurements and their uncertainties, leading to an improved representation of the line strokes in the final simplified drawings.

In 2016, the algorithm presented by JD. Favreau *et al.* [12] aims to balance fidelity (i.e. the parametric curves should closely approximate the input drawing) with simplicity to generate an accurate and compact number of curves. This algorithm begins by extracting a single-pixel width skeleton from the image representing line centers and junctions. This data is transformed into a graph where edges symbolize the line segments and vertices denote junctions, endpoints, and turns. However, directly fitting curves to these edges often result in over-segmentation due to junctions breaking long curves and thick lines causing additional branching in the skeleton. To resolve this issue, the algorithm employs a global optimization strategy to minimize curve count and fitting errors simultaneously. It groups successive edges that can be represented by a single curve without sacrificing accuracy, introducing a novel representation using hypergraphs. This allows multiple intersecting curves to share edges, crucial in simplifying the curve network and resolving issues like extraneous branching at junctions.

More recently, P. Najgebauer *et al.* [13] introduced a method that performs fast vectorization of line drawings based on a multi-scale second derivative detector and inertia-based line tracing to improve accuracy at the junctions significantly. The multi-scale second derivative detector is used to identify edges and details within the grayscale sketch by using a technique known as the summed-area table. This table speeds up the process of detec-

tion of lines in the sketch drawings by quickly calculating the total pixel values in specific regions, making the detection process more efficient. They applied ”inertia” to mimic the natural movement of the person sketching. By mimicking these movements, their method aims to improve the handling of junctions.

Some vectorization techniques have matured enough to be integrated in commercial tools, such as Potrace Inkscape [5] (explained in detail in Section 4.2), Adobe Illustrator Image Tracing [14], and CorelDraw [15]. These semi-automatic tools focus on maintaining high visual accuracy and quality during the vectorization process.

2.1.2 Deep Learning Approaches

All the above-discussed methods focus on improving geometric primitives and visual accuracy for vectorization. Adopting completely different strategies, [16], and [17] introduced instance segmentation techniques that are entirely data-driven and utilize neural networks to segment strokes based on the global context of the image. N. Inoue *et al.* [16] used a model based on MaskRCNN [18] to perform instance segmentation of hand drawings. MaskRCNN comprises two stages. The first stage, known as the Region Proposal Network (RPN), identifies the regions that are likely to contain an object and proposes bounding boxes around these regions. These bounding boxes are called Region of Interest (RoI). The second stage comprises a RoIAlign layer to extract features from each RoI candidate. These features are passed through three distinct branches: classification, box, and mask branches to classify, predict, and generate binary masks for each RoI candidate, respectively. The authors proposed two key modifications to the MaskRCNN architecture. Firstly, they upsampled the masks generated from the mask branch of the MaskRCNN. This is because in the line drawings, instances to be detected are smaller and thinner as compared to other images, and the output of the general MaskRCNN is not able to capture the finer details of the instances. Secondly, a post-processing step is applied to correct the mismatch between

the pixels. The model filters the predictions on the background and assigns the correct label to any pixel that does not belong to an instance by assigning the nearest neighbor using the nearest neighbor search technique.

B. Kim *et al*. [17] introduced a pair of neural networks, named PathNet and OverlapNet, to identify common patterns and overall relationships among the paths. PathNet is used to find the correct path by measuring a path similarity term, utilized to assess the likelihood of two pixels belonging to the same path. The PathNet neural network comprises 20 sequential blocks, with each block having 64 convolution filters of size 3x3. Each convolution block is followed by batch normalization and Rectified Linear Unit (ReLU). PathNet takes two-channel input, comprising the input image and a mask image, which is black except at a given indicator pixel p ($p \in P^{'}$, where $P^{'}$ is a set of pixels that represents a set of all path pixels, excluding the duplicate overlapping pixels). The output of the network is a single channel image, with values in the range of 0 to 1, giving the confidence score of the likelihood of p lying on the same path as compared to all other pixels in the image. Figure 2.1 illustrates a sample image that shows the relationship between two pixels, one lying on the same path and the other on a different path. All the traditional pixel-labeling methods consider that each pixel in an image corresponds to a unique label. However, there are overlapping regions where multiple labels need to be assigned to a single pixel. Therefore, the authors introduced a second network named OverlapNet, trained to provide semantic clues to help resolve ambiguities in the overlapping regions. Its architecture is similar to that of PathNet; however, instead of using a two-channel input, this network utilizes a single-channel input containing an input image, and a sigmoid function instead of ReLU. A threshold is applied to the output to obtain a binary map, indicating the region where overlap occurs. Based on the overlap map, new pixels corresponding to the overlapping pixels are added to $P^{'}$. This algorithm is limited to accounting only for the cases where two paths overlap on a single pixel. Finally, the predictions from the above models are

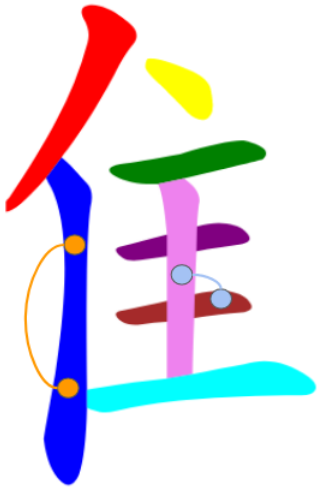


Figure 2.1: Sample image illustrating relationship between two pixels: yellow dot represents pixel p and q lying on the same path (i.e., $l_p = l_q$), whereas the blue dot illustrates the pixels on a different path (i.e., $l_p \neq l_q$), where l_p: label at pixel p and l_q: label at pixel q.

combined by defining a global energy function using Markov Random Field (MRF). This global energy function is optimized via graph cuts to handle multi-label problems.

We consider [17] as the state-of-the-art approach in vectorization viewed as an instance segmentation problem. B. Kim *et al.* [17] introduced an effective approach that performs really well in terms of testing accuracy. However, their process of iterative optimization of the energy is time-consuming. In comparison, the fast instance segmentation technique proposed by N. Inoue *et al.* [16] exhibits faster processing times than [17]. While there is a trade-off between speed and quality of the results, B. Kim *et al.* approach [17] is highly effective and superior in terms of accuracy. Therefore, we have chosen [17] as our baseline state-of-the-art method. Our approach proposes a different way of solving ambiguities generated by multi-path overlap, based on Single-Class Instance Segmentation (SCIS). Overall, our method is significantly faster than [17] and also compares favorably in terms of accuracy.

2.2 Loss Function

The selection of a proper loss function to perform segmentation is crucial for the overall performance of the segmentation model. In this section, we review various loss functions that have been developed to perform various segmentation tasks.

The cross-entropy loss function introduced by M. Yi-de [19] is one of the primary choices for segmentation. It measures the dissimilarity between the ground truth and the predicted probabilities. This loss function is widely used in classification and segmentation tasks because of its ability to resolve problems associated with class imbalance, unique targets, complex regions, and performance optimizations. To improve the ability of a model to classify or segment targets precisely, this function enables the customization of the learning process. Cross entropy is calculated by element-wise multiplying the ground truth (one-hot encoded, i.e., representing each class using binary vectors, where the element corresponding to the actual class is set to 1, and all others are set to 0) with the predicted probabilities and summing them over all classes. The cross-entropy loss function is given below:

$$H(x, y) = -\sum_{N \in classes} x(N) \log y(N) \tag{2.1}$$

where $x(N)$ is the true probability distribution and $y(N)$ is the predicted probability distribution and N represents classes. Several loss functions have been derived from the concept of cross entropy: binary cross-entropy, weighted binary cross-entropy [20], balanced cross-entropy [21], sparse categorical cross-entropy loss [22], label smoothing loss [23] and focal loss [24]. The binary cross-entropy loss function measures the dissimilarity between the true binary labels, which can be 0 or 1, and the binary predicted probabilities. This type of cross-entropy loss function is commonly used for binary classification tasks, aiming to categorize data into two distinct classes. V. Pihur [20] introduced the weighted binary cross-entropy loss function specifically designed to mitigate challenges related to class im-

balances in a dataset. This loss function is a variant of binary cross-entropy and assigns weight to the positive class using a specified coefficient. This coefficient is used to tune both False Positive (FP) and False Negative (FN), i.e., if the coefficient > 1, the number of FN decreases, whereas if the coefficient < 1, the number of FP reduces. S. Xie [21] proposed the balanced cross-entropy loss designed for the same task, i.e. to address the class imbalances. Unlike weighted cross-entropy, which focuses solely on weighting positive examples, this loss function extends the weighting to both positive and negative examples. This loss function automatically balances the loss between positive and negative classes by adjusting the weights based on the inverse of class frequency. The sparse categorical cross-entropy loss function by BN Chaithanya [22] is similar to the standard cross-entropy loss function, used to handle multi-class classification problems. The key difference lies in the handling of true labels. Instead of using one-hot encoded true labels, the sparse categorical cross-entropy loss function is designed to work directly with true integer labels. This characteristic makes it widely useful when dealing with a large number of classes. TY Lin [24] used the focal loss function to address challenges associated with class imbalances, especially foreground/background imbalances, by directing attention to difficult classes with low probabilities. This is achieved by reducing the weight applied to the well-classified samples.

Intersection over Union (IoU)-based loss functions encourage models to minimize the dissimilarity between the ground truth and the predicted results while maximizing their similarity. One of the standard IoU-based loss functions used to quantify the degree of overlap between regions is Dice Loss (discussed in detail in Section 3.3). The Dice loss [25] is a metric commonly used to assess the quality of segmentation results by maximizing the overlap between ground truth and predicted masks, is derived from the Dice coefficient [26]. This loss function is particularly suitable for performing segmentation tasks with highly imbalanced classes. Several variants derived from Dice loss are Jaccard loss, Tver-

sky loss [27], and lovasz-softmax loss [28]. Jaccard loss is the complement of the Jaccard index (IoU), i.e. the ratio of the intersection of the true mask with the predicted mask to the total area covered by both masks. The Tversky loss utilizes the Tversky index, which is the generalization of the Dice coefficient and Jaccard index. It offers adaptability in handling class imbalance by incorporating adjustable parameters to regulate the influence of false positives and false negatives. Lovasz-softmax [28] is used for multi-class semantic segmentation by optimizing the mean IoU loss. Instead of using the Jaccard index directly, this loss substitutes it with a manageable convex surrogate. A manageable convex surrogate is a simplified and convex alternative to a more complex loss function, that aims to maintain convexity (i.e., to ensure convergence to a global minimum) to simplify optimization in a more efficient way.

Margin-based loss functions such as hinge loss [29] and triplet loss [30] optimize the model's performance around the decision boundaries. They penalize the error near the decision boundary that separates classes. The hinge loss [29] (discussed in detail in Section 3.3) is used for binary classification tasks to maximize the distance between the data points. The triplet loss function [30] uses a reference input or anchor, where the aim is to minimize the distance between the anchor and positive examples while maximizing the distance between the anchor and negative examples.

Different loss functions with different objectives can be combined into a single composite loss function, known as compounded loss. The main objective of combining the losses is to incorporate multiple aspects of a problem and criteria into the training process. Combo loss [31] is a weighted sum of the cross entropy and Dice loss. The Label-Distribution-Aware Margin (LDAM) loss [32] function is designed to handle heavy class imbalance by assigning larger margins to minority classes. This loss function is based on the soft margin loss function which allows for a certain level of misclassification while keeping the margins as wide as possible to ensure correct classification of other points. It serves as a multi-class

extension of hinge loss.

All the above-discussed loss functions have different aspects and a specific loss function can be selected based on the training objective. Based on these loss functions, we proposed a novel loss function to address the inherent imbalance in our specific use case, characterized by a skewed distribution and following perceptual grouping principles (similarity, continuity, and closure). Our loss function can account for the spatial aspects of the image while optimizing minority labels.

Chapter 3

Proposed Approach

In this chapter, we present a deep-learning-based model to perform Single-Class Instance Segmentation (SCIS) (a special case of instance segmentation where there is only one class and the goal is to segment each instance present within an image). We have one class known as foreground, and instance segmentation is performed on this class by segmenting each individual stroke in the line art drawing and assigning unique labels to each stroke instance. We propose a new architecture, named Multi-Focus Attention UNet (MFAU) and inspired by Attention UNet [3], to perform SCIS. This architecture is designed to achieve precise and accurate stroke segmentation of the input image, demonstrating superior performance compared to state-of-the-art methods across various performance metrics. The novelties included in our model consist of Multi-Focus Attention Gates (MFAG) (Section 3.2.1), Highway Skip Connections (Section 3.2.2), and Margin Regularized Loss Function (Section 3.3). This chapter starts by discussing the deep-learning models that we considered when proposing our architecture. Section 3.2 and Section 3.3 discuss in depth our proposed segmentation model and the three main contributions of our approach.

3.1 Motivation for our architecture

This section discusses the progressive developments that took place for semantic segmentation and the motivation that led to the development of our proposed architecture.

In 2015, Fully Convolutional Networks (FCN) were introduced by Long *et al.* [1] to perform per-pixel classification by combining convolution and upsampling operations. Figure 3.1 shows the FCN architecture. The combination of convolution (extracting different features from an image by applying different filters, where each filter highlights specific features within an image) and pooling operations (reducing the dimensions of the feature maps obtained from the convolution layer to extract essential features) are used in FCN to downsample the image by reducing its spatial resolution while extracting complex patterns and fine details from the image. Once the local patterns (i.e., edges, textures and other visual patterns) are extracted, a transposed convolution operation is applied to upsample the feature maps, therefore producing a dense output map of the same size as the input image. The segmentation output is coarser when using a pixel stride of 32 (in the architecture known as FCN-32) for upsampling, as due to the large stride, fine details that got lost during downsampling are not recovered. Therefore, *skip connections* are introduced to combine the upsampled feature maps with higher-level information. Skip connections, also known as shortcut connections and initially introduced in ResNet [33] to improve image classification results by forwarding the output from one layer of the network as input to another while skipping some layers in between. Figure 3.2 illustrates the block diagram of an FCN network using skip connections. In FCN-16, the 2x upsampled predictions of the final layer (pool 5) are combined with the predictions from the previous pooling layer (pool 4). Finally, the 16-pixel stride is applied to upsample the predictions back to the pixels, preserving the finer details. In FCN-8, the predictions from pool 3 are combined with the 2x upsampled predictions from the previous fused input of FCN-16 and then upsampled

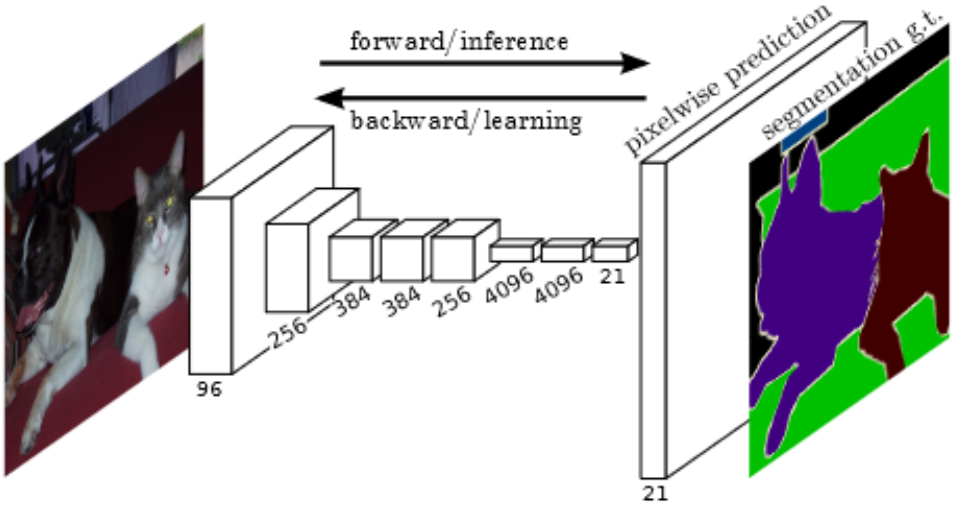


Figure 3.1: Fully Convolutional Networks (FCN) [1] performing downsampling and upsampling of the input image to obtain a dense (pixel-wise) prediction output map.

at stride 8 to obtain more precise predictions. Therefore, this approach improves the output predictions as compared to other FCNs (FCN-32 and FCN-16).

As an extension to FCN, UNet was introduced by Ronneberger *et al.* [2] to obtain more precise segmentation results comared to FCN. Figure 3.3 shows that UNet consists of a contracting path (encoder), bottleneck, an expanding path (decoder), skip connections, and a final 1x1x1 convolution layer, forming a U-shaped architecture. The contracting path is designed like a typical convolutional network that uses filters to extract spatially coherent features, where each layer consists of convolution, ReLU, and max pooling operations. The spatial dimensions are gradually downsampled while increasing the feature dimensions to provide a larger receptive field. The contracting path is responsible for capturing higher-level features and extracting various patterns and structures in the input image at different levels of complexity. The bottleneck acts as the bridge between the contracting and expanding paths. After the encoder has reduced the spatial dimension, the bottleneck plays a crucial role in providing a controlled way first to compress and then expand these features. It acts as a balance between spatial and feature representation, capturing both the local and global context of the input image. The decoder operates in reverse fashion with respect

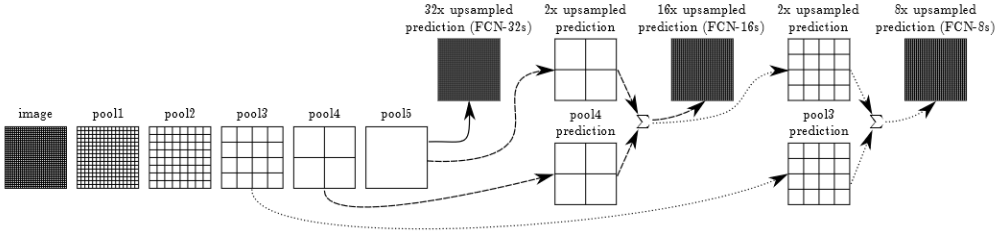


Figure 3.2: Fully Convolutional Networks (FCN) implemented with skip connections [1], to combine higher-level and lower-level information. The relative spatial coarseness is shown in the form of a grid. To keep it simple, only the pooling and prediction layers are included. FCN-32 (solid line) upsamples the predictions from pool 5 at stride 32 back to an image. FCN-16 (dashed lines) combines the predictions from the final layer and pool 4 at stride 16. FCN-8 (dotted lines) combines the predictions from the previous two layers with pool3 at stride 8.

to the encoder. It consists of a transposed convolutional layers that upsample the spatial dimension and reduce the feature channels for accurate dense classifications with dimensions closer to the input image. The coarser details obtained from the contracting path are concatenated with finer details of the expanding path using *skip connections* to localize the object correctly. In the final layer, a 1x1x1 convolution operation is applied to map the decoder layer feature vectors to an N-d (where N is the number of classes) feature vector probability map. Due to the symmetric structure and the presence of skip connections at each layer, UNet performs well even with a limited number of training images as it is able to preserve the higher resolution that may be lost during the downsampling and finer details that may be lost during the upsampling of the input image, in contrast to FCN.

To improve the segmentation results further by enhancing the model's ability to capture salient regions that might not be detected by the standard UNet, Attention Gates (AG) were added by Oktay *et al*. [3] in the decoder phase of the UNet architecture. Figure 3.4 illustrates the block diagram of Attention UNet. Attention mechanisms are added before concatenating the information from the encoder phase with the decoder phase at each decoding level. These gates control the flow of information by learning to adapt to relevant spatial regions, thus improving overall segmentation results. As can be seen from Figure

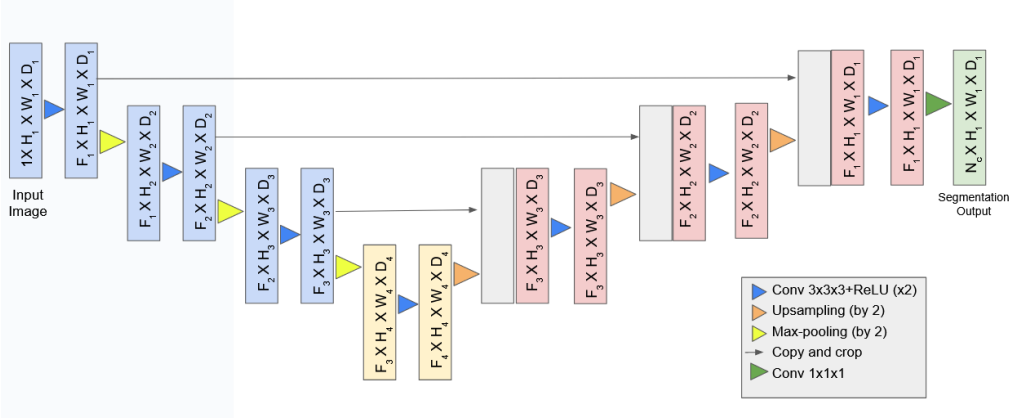


Figure 3.3: Block diagram of UNet architecture (modified from [2]). The input image is downsampled in the encoder (shown in blue boxes) and upsampled in the decoder path (shown in red boxes). The bottleneck (shown in yellow boxes) acts as a bridge between the encoder and decoder. White boxes depict the feature maps copied from the encoder. The final layer is shown in the green box. The $FxHxWxD$ notation represents the multi-channel feature maps where F depicts the number of channels, $HxWxD$ are the *heightxwidthxdepth* of the image. N_c in the final layer represents the number of classes.

3.6, the gating signal collected at a coarser scale of the decoder containing contextual information and input feature maps from the encoding layer is passed through the attention mechanism to determine attention scores, $\alpha \in [0, 1]$. These attention scores represent the importance of each spatial location of the feature maps. The input feature maps containing higher-level spatial information are pixel-wise multiplied by the attention scores to enhance the relevant spatial location in the feature map. This forces the network to focus on critical spatial regions while suppressing the feature responses from the irrelevant regions, including background and other noisy information. By suppressing these activations and learning to focus on specific regions, the overall representational complexity of the network is improved, along with the ability to capture salient objects in the image.

Due to the concatenation of information at each stage, Attention UNet suffers from high computational complexity and a large number of model parameters. Our architecture presents a trade-off between high model complexity and improved precision. To improve the segmentation results further, along with reducing model complexity and the number of

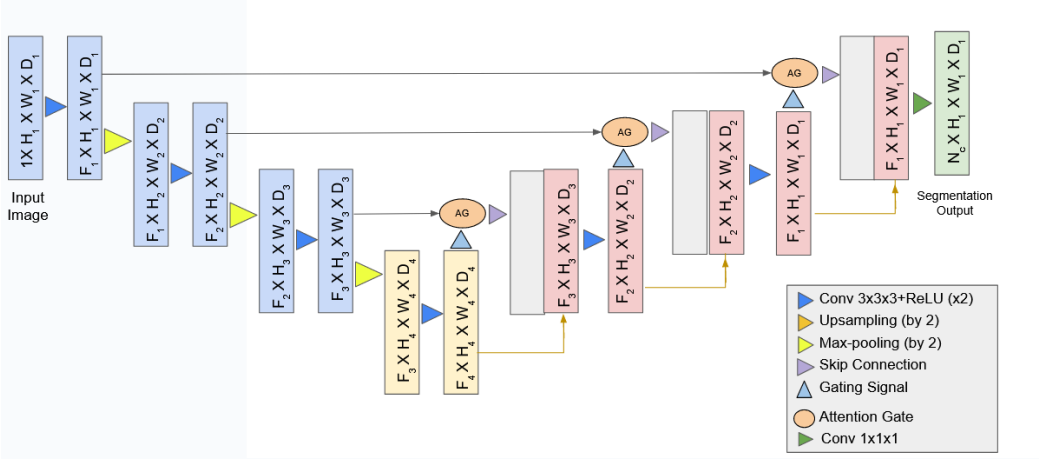


Figure 3.4: Block diagram of Attention UNet Model (modified from [3]). The input image is downsampled in the encoder (shown in blue boxes) and upsampled in the decoder path (shown in red boxes). The bottleneck (shown in yellow boxes) acts as a bridge between the encoder and decoder. White boxes depict the feature maps copied from the encoder. Features from the encoder and decoder are passed through the attention gates and concatenated with the corresponding layer in the decoder. The final layer is shown in the green box. The $FxHxWxD$ notation represents the multi-channel feature maps where F depicts the number of channels, $HxWxD$ are the *heightxwidthxdepth* of the image. N_c in the final layer represents the number of classes.

parameters, we propose a novel architecture, Multi-Focus Attention UNet (MFAU) (shown in Figure 3.5). This architecture is similar to the Attention UNet (shown in Figure 3.4), however, the *skip connections* (shown as white boxes in Figure 3.4) that perform a concatenation of global and local information are replaced with *Highway Skip Connections* (introduced by Srivastava *et al.* [34][35] and shown as red dotted arrows in Figure 3.5). Highway Skip Connections (discussed in detail in Section 3.2.2) are in fact another type of attention, called ”branch attention” [36] (as opposed to spatial attention in Attention UNet). These connections consist of gating mechanisms (shown in Figure 3.9) that allow only selective information to pass through the network and reduce the overall model complexity. While there is a slight degradation in the segmentation results compared to those achieved with Attention UNet, the model complexity has significantly improved when replacing *skip connections* with the *Highway Skip Connections* (a quantitative comparison

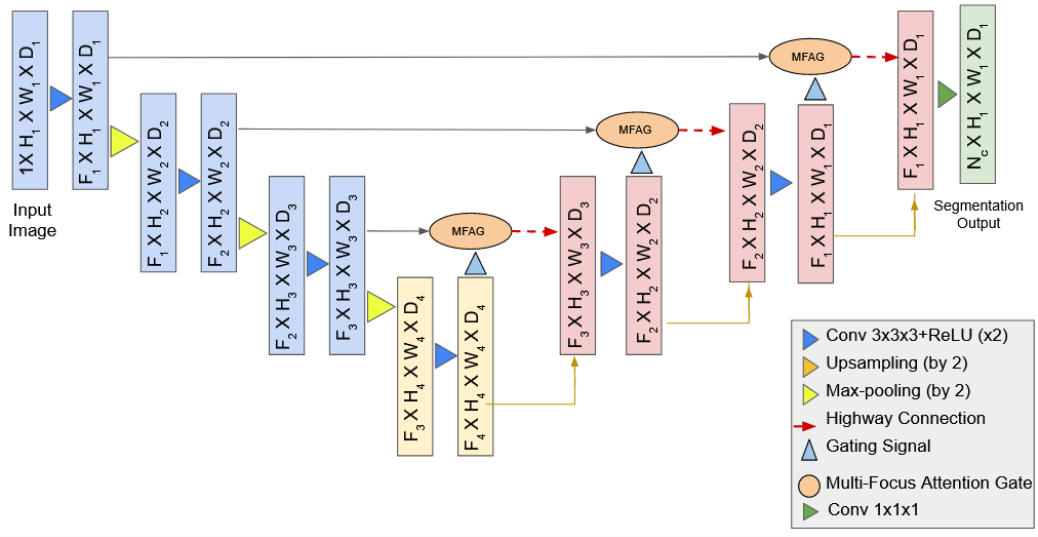


Figure 3.5: Block diagram of our proposed Multi-Focus Attention UNet (MFAU) Model. The input image is downsampled in the encoder (shown in blue boxes) and upsampled in the decoder path (shown in red boxes). The bottleneck (shown in yellow boxes) acts as a bridge between the encoder and decoder. This concept is similar to Attention UNet [3]; the difference lies in the features from the encoder and decoder that are passed through the MFAG (the AG in Figure 3.4 is replaced with MFAG and shown as orange ellipses) and further through the *Highway Skip Connections* (the *skip connections* that performs concatenation of information from the encoding and decoding layer, shown in white boxes in Figure 3.4 are replaced with the red arrow representing *Highway Skip Connections*). The final layer is shown in the green box. The $FxHxWxD$ notation represents the multi-channel feature maps where F depicts the number of channels, $HxWxD$ are the *heightxwidthxdepth* of the image. N_c in the final layer represents the number of classes (stroke instance labels in our case).

is shown in Section 4.2). Additionally, we introduce *Multi-Focus Attention Gates (MFAG)* (discussed in detail in Section 3.2.1 and shown in Figure 3.7) that replace *Attention Gates (AG)* to make the model adapt to the relevant regions by amplifying important features further, facilitating the selection of the most pertinent information while diminishing the impact of irrelevant and less significant elements. Thus, the segmentation results are improved compared to those obtained using Attention UNet (the quantitative comparison of these results is shown in Section 4.3).

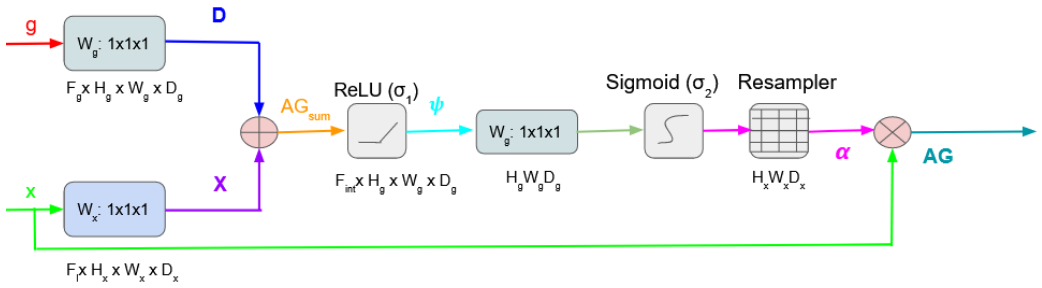
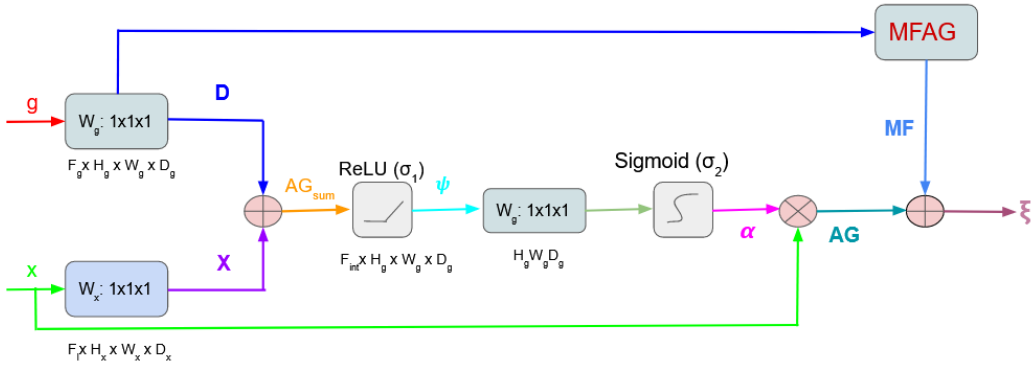


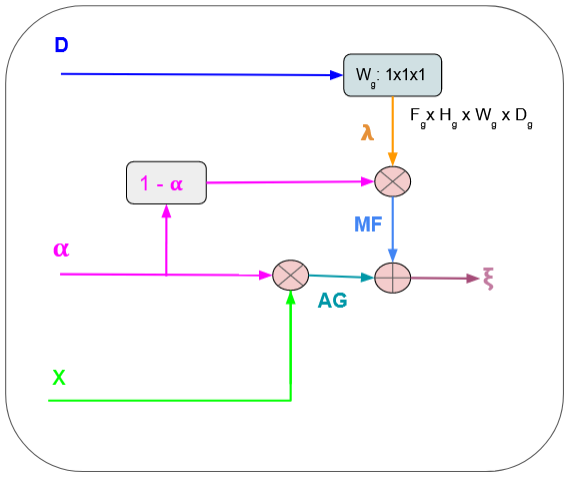
Figure 3.6: Block Diagram of Attention Gate (modified from [3]). Attention coefficient α, is computed using the input feature (x) from the encoder and gating signal (g) from the decoder. These attention coefficients are used to scale input features by performing element-wise multiplication between them. Different colors show the different flow of information. The output of the attention gate is represented as AG. The $W_g : 1x1x1$ in the boxes represents the $1x1x1$ convolution operation.

3.2 Proposed Architecture

Figure 3.5 illustrates the detailed architecture of our proposed approach. Although our aim is to perform instance segmentation, we have based our approach on the semantic segmentation models. Most of the instance segmentation models that provide precise segmentation results, such as YOLACT [37] and MaskRCNN [18], are two-stage-segmentation models that can be slow in terms of inference speed, and implementing these models is computationally expensive as they require advanced hardware and resources for training. Also, Inoue *et al.* [16] had already implemented MaskRCNN to perform instance segmentation of the line art drawings vectorization. However, despite our efforts to reproduce their results, we were not able to achieve segmentation results close to the ones that they reported, nor were these results better than the state-of-the-art method. Furthermore, as we have a single class, the multi-channel output of the semantic segmentation models can be used to predict per-pixel labels, with each channel representing an individual instance of a stroke instead of a class. Since our objective of SCIS can be fulfilled through lightweight, faster-to-implement semantic segmentation models rather than instance segmentation models (we



(a) Block Diagram of our proposed Multi-Focus Attention Gates (MFAG).



(b) MFAG block

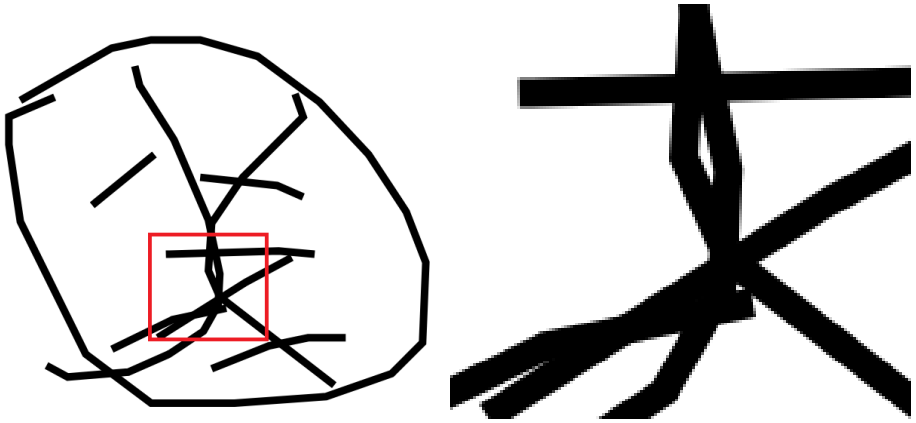
Figure 3.7: a) Block Diagram of the Multi-Focus Attention Gates (MFAG). b) MFAG block. Attention coefficient α, is computed using the input feature (x) from the encoder and gating signal (g) from the decoder. These attention coefficients are used to scale input features by performing element-wise multiplication between them. Different colors show the different flow of information. The output of the attention gate is represented as AG. The W_g : $1x1x1$ in the boxes represents the $1x1x1$ convolution operation. The output of the MFAG block, represented as MF is concatenated with AG to get the final output (ξ). This process enhances the scaled input features by incorporating the gating signal, further improving the focus on specific regions.

hypothesize that each stroke is distinct enough for a semantic segmentation model to perceive it as belonging to a separate class), our proposed architecture is based on a semantic segmentation model, named Multi-Focus Attention UNet (MFAU).

The essential components of our architecture are: *encoder*, *bottleneck*, *decoder*, *multi-focus attention gates*, *Highway Skip Connections*, and *final layer*. The *encoder*, *bottleneck*, *decoder*, and *final layer* blocks are identical to the ones used in Attention UNet. Detailed explanations of our *multi-focus attention gates* (3.2.1) and *Highway Skip Connections* (3.2.2) are given below.

3.2.1 Multi-Focus Attention Gates (MFAG)

Figure 3.7 represents the block diagram of *MFAG* (the MFAG block represents the modifications we made to the AG). The detailed illustration of the MFAG block is given in Figure 3.7b). As can be seen from comparing Figure 3.6 and Figure 3.7, until the sigmoid function, both *MFAG* and *AG* are performing the exact same task. In the final step of *AG*, the attention coefficients $\alpha \in [0, 1]$ are calculated to highlight each spatial region of the input feature vector from the encoder (by multiplying α with the input feature vector x). Figure 3.8 shows a sample input image containing overlapping regions. By looking at the zoomed-in image (Figure 3.8b), one may note how difficult it is to attribute labels to strokes. As a result, the global context, which refers to the comprehensive understanding of the entire image, becomes crucial to determining the relationship between individual strokes. By incorporating our modifications, the network gains the ability to encode this contextual information by utilizing the gating feature vector, which is collected at a coarser scale of the decoding layer. This is achieved by utilizing attention coefficients, which involve multiplying their complement (1 - α) with the convoluted gating feature vector λ. Thus, based on the attention weights, *MFAG* controls the amount of higher-level contextual information (from the gating signal) and lower-level information (from the input feature



(a) (b)

Figure 3.8: (a) Example of an input image from our dataset with red highlighted region zoomed in. (b) The zoomed-in version of the input image depicts the overlapping regions, emphasizing the importance of considering the global context of the overall image to accurately identify which label belongs to each stroke.

vector) that need to be combined to further enhance relevant semantic details of the input image and discard the irrelevant ones (explained in Equation 3.7).

Let the inputs to our proposed *MFAG* architecture (as shown in Figure 3.7a) at layer l from the decoding layer be represented as g^l, and the input from the encoding layer be represented as x^l. The gating feature vector g^l is used to ascertain the regions of focus and incorporate higher-level contextual information. To perform sequential linear transformations and extract the local and global context of the image, a convolution followed by batch normalization is applied to the gating and local feature vectors at each layer. The resultant transformed feature vectors are represented as $D(g^l, W_g^l)$ and $X(x^l, W_x^l)$, and parameterized by W_g^l and W_x^l for gating and local features respectively. For simplicity, the transformed feature vectors $D(g^l, W_g^l)$ and $X(x^l, W_x^l)$ will be used in the equations as $\mathbf{D}^l$ and $\mathbf{X}^l$. These transformed feature vectors are then summed, given as:

$$AG_{sum}{}^l = \mathbf{D}^l + \mathbf{X}^l \tag{3.1}$$

Following this summation, a non-linear activation function σ_1 is applied to the resultant

output of the Equation 3.1. The authors of Attention UNet [3] have used the ReLU as a non-linear activation function. To maintain consistency, we also use ReLU. The following equation shows the non-linear activation being applied:

$$\psi^l = \sigma_1(AG_{sum}{}^l) = max(0, AG_{sum}{}^l) \tag{3.2}$$

where ψ^l represents the intermediate coefficients. A linear transformation is applied on ψ using 1x1x1 convolutions , utilized to downsample the input feature maps to the gating feature vector dimensions. A non-linear activation function, i.e., sigmoid (represented as σ_2), is applied to the linearly transformed results. The sigmoid activation function has been used by the authors to yield sparser activations instead of a softmax function that normalizes the attention coefficients, thus resulting in better training convergence in Attention Gates (experimentally proven in [3]).

$$\alpha^l = \sigma_2(\psi^l) = \frac{1}{1 + e^{-\psi^l}} \tag{3.3}$$

where α represents the attention coefficients $\in [0, 1]$, which highlight feature responses from the salient regions and suppress the ones with lesser semantic values. In AG, a grid resampler is applied to the attention coefficients that are implemented using trilinear interpolation to make the dimensions of the feature maps similar to the dimensions of the input feature vector to multiply them element-wise. In our case, the dimensions of the attention coefficient feature maps happen to be similar to those of the input feature vector x. Therefore, for the purpose of simplicity, we omit this step. The element-wise product of attention coefficients with the input feature vector from the encoding stage (i.e., used to scale each spatial location of the input feature vector based on the attention scores $\alpha \in [0, 1]$), is given as:

$$AG^l = \alpha^l . x^l \tag{3.4}$$

To perform the multi-focus attention mechanism (shown in Figure 3.7), $\mathbf{D}^l$ is sequentially transformed again. This transformation serves the purpose of further enhancing the gating features at a coarser scale through the utilization of 1x1x1 convolution operations, denoted as $H(g^l, W_h^l)$ and parameterized by W_h^l for multi-focus feature vectors ($H(g^l, W_h^l)$ is represented as λ^l for simplicity). This transformed gating feature vector is element-wise multiplied with the complement of attention coefficients to further improve the focusing ability of the network, select the most relevant features, and discard the less important ones.

$$MF^l = (1 - \alpha^l).\lambda^l \tag{3.5}$$

The final output of the MFAG is given by concatenating Equation 3.4 with Equation 3.5:

$$\xi^l = AG^l + MF^l \tag{3.6}$$

$$\xi^l = \alpha^l . x^l + (1 - \alpha^l).\lambda^l \tag{3.7}$$

where ξ^l represents the selection feature vectors at layer l. These dimensions will always be the same as the dimensions in the input feature vector. The attention coefficient $\alpha \in [0, 1]$ acts as a weighting mechanism that controls the amount of focus to be given to the input feature map from the encoding stage or to the gating feature map from a coarser decoding layer. If α is close to 0, the output of the *MFAG* will focus more on the spatial information, whereas if α is close to 1, the output will focus more on the contextual information from

the input image.

Based on Equation 3.7, a special case of saturation of the attention coefficients $\alpha \in [0, 1]$ is given below:

$$\xi^l = \begin{cases} x, & \text{if } \alpha = 0 \\ \lambda, & \text{if } \alpha = 1 \end{cases} \tag{3.8}$$

For simplicity, the symbol for layer l has been omitted. If α is 0, the input feature vectors from the encoding stage are highlighted, whereas if α is 1, the gating feature vectors from the decoder layer collected at a coarser scale undergo a linear transformation that helps to recognize complex patterns of the feature map and highlight these patterns.

3.2.2 Highway Skip Connections

Highway networks were introduced in 2015 by Srivastava *et al.* [34]. There is plenty of evidence related to how the depth of a neural network improves its performance; however, optimizing a deeper network is a challenging task. Highway networks were explicitly designed to overcome these challenges and train very deep neural networks. They introduce adaptive gating units (shown in Figure 3.9), a component that adaptively learns to control the flow of information through the network. As a result, there can be paths through which information can traverse directly several layers without being altered. These adaptive gating units (known as Highway Skip Connections) are used at each layer in the Highway networks and were inspired from the gating mechanism used in long short-term memory (LSTM) networks [38]. The main goal of these gating functions is to learn to select whether the information needs to be passed or transformed through the network. This way, the network is able to dynamically learn which features are relevant to a particular problem and adapt accordingly. According to the best of our knowledge, we are the first group to use the concept of highway networks for a small number of layers and integrate it with a

UNet-based model.

Figure 3.9 illustrates a detailed block diagram of Highway Skip Connections. The output of the MFAG, represented as ξ^l, acts as an input to Highway Skip Connections at each decoding layer l. Two sets of operations are applied on the input feature vector ξ^l: a *transformation gate* and a *transformation layer* (the difference between these two operations is the use of different non-linear activation functions). These operations are defined as $G(\xi^l, W_G^l)$ and $T(\xi^l, W_T^l)$, parameterized by W_G^l and W_T^l at every layer l for the transformation gate and transformed input, respectively. The *transformation gate* consists of a 1x1x1 convolution operation and a sigmoid function. This sigmoid function produces a value in the range 0 and 1, depicting how much of the transformed input should be passed through the networks. If the value is close to 1, the input flows through the layer, whereas if the value is close to 0, most of the inputs get transformed. The *transformation layer* consists of a 1x1x1 convolution operation followed by a ReLU function to apply a non-linear transformation on the input image. This transformation enables the network to capture complex features from the input image. The two different non-linear activation functions are used in a certain way to mitigate the vanishing gradient problem, as even if the transformation gate collapses due to saturation towards 0, the input is still passed through the network, and if the saturation of the transformation gate is towards 1, the input gets fully transformed (this special case is shown in Equation 3.13).

As seen from Figure 3.9, information is passed through two information paths (the paths are shown in red and green color). In the first path (shown in red color), the input feature vector ξ^l is directly element-wise multiplied with the complement of the *transformation gate* feature maps, represented as $1 - G(\xi^l, W_G^l)$ (henceforth written as $(1 - G)^l$ for simplicity). This element-wise product for the first information path is given as:

$$TH_1^l = (1 - G)^l . \xi^l \tag{3.9}$$

In the second information path (shown in green color), the feature vectors from both the *transformation gate* $G(\xi^l, W_G^l)$ (represented by G) and *transformation layer* $T(\xi^l, W_T^l)$ (represented by T) are element-wise multiplied by each other at each layer l.

$$TH_2{}^l = G^l.T^l \tag{3.10}$$

Highway Skip Connections are formed by combining information paths constructed in Equations 3.9 and 3.10. Therefore, the overall Highway Skip Connections are represented as:

$$H^l = TH_1{}^l + TH_2{}^l \tag{3.11}$$

$$H^l = G^l.T^l + (1 - G)^l.\xi^l \tag{3.12}$$

where H symbolizes the Highway Skip Connections output at each decoding layer l. The dimensions of H, G, T and ξ will always be the same.

Based on equation 3.12, the special case of saturation of the transformation gate can be given by the following two conditions:

$$H = \begin{cases} \xi, & \text{if } G = 0 \\ T, & \text{if } G = 1 \end{cases} \tag{3.13}$$

For simplicity, the symbol for layer l has been omitted. Therefore, depending on the value of the transformation gate, Highway Skip Connections control the flow of information by selecting whether to emphasize the transformed input or the original input.

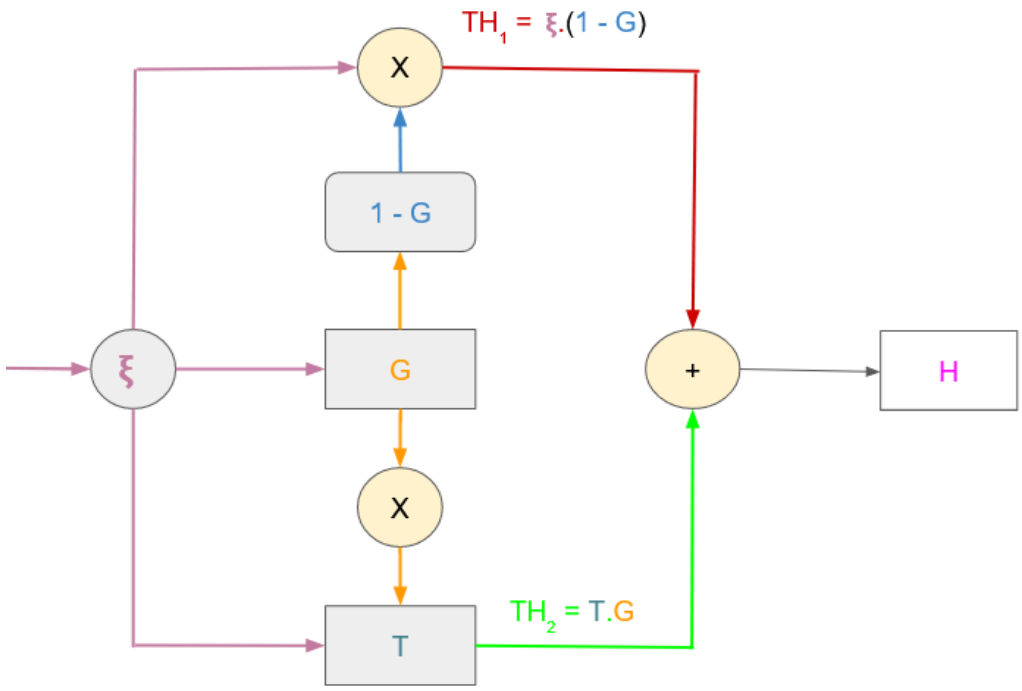


Figure 3.9: Block diagram of Highway Skip Connections. ξ represents the input from the encoding stage, while H represents the output of the Highway Skip Connections, and G and T are the transformation gate and transformation layer, respectively. The red and green paths show the two paths from where the information flows. The 'x' and '+' symbols denote element-wise multiplication and addition, respectively.

3.3 Compounded Loss Function

We have a specific use case that is inherently imbalanced in a heavy-tailed distribution manner. The output of the segmentation model consists of a static number of channels (the value is fixed as 128), and the average number of instances in our problem domain is 9 (see Section 4.1). As a result, we observe a heavy distribution until 9 while after that there is a decreasing trend. For example, if we have an image with 5 strokes and another with 10 strokes, the distribution is concentrated with a heavy frequency of instances up to 5, gradually decreasing up to 10 strokes.

The output of the proposed instance segmentation model is a static number of channels,

which makes this problem analogous to a multi-label classification problem. However, unlike a multi-label classification problem where each channel represents a specific class prototype, in our use case, we do not have any fixed class prototype, i.e., any instance of the stroke can appear in any channel. Figure 3.10 illustrates an example where similar strokes will not necessarily appear in the same channel for both images. As can be seen in the figure, the strokes appearing in the first and second channels for the top image appear in the second and third channels for the bottom image. Moreover, different artists have different drawing styles. As a result, the same stroke can represent different instances. Figure 3.11 illustrates different drawing styles for a cat; most of the time, a cat is drawn with two ears, a face, and each whisker separately labeled, however, there are cases where the two ears are labeled as a single entity, or the whole face is labeled as one. The skewed distribution in our use case means that infrequent logit channels may suffer from floating-point truncation. This occurs when the values in the channels become very small and are eventually rounded to zero due to limited precision in floating-point representation. We mitigate this issue using margin scaling of the logits. By scaling the margin, the network becomes more tolerant to small variations in the logits. This prevents dead neurons which allows for the full power of the attention gates to be used to model the pattern variability. Therefore, we propose a novel loss function named Margin-Regularised Loss Function (MRLF) that addresses this variation and identifies the individual instances accurately.

Apart from assigning correct labels to each stroke instance, we also need to consider the spatial information of the strokes to incorporate the precise positioning, overlapping, and relationship between these strokes. The spatial imbalances favor instances with relatively larger spatial strokes (strokes that occupy more pixels) in the dataset to the detriment of smaller ones. To address spatial imbalance issues, Dice Loss is used to assess how well the ground truth and predicted output overlap with each other. It gives equal weights to pixel-wise true positives and false negatives, encouraging the model to capture the spatial

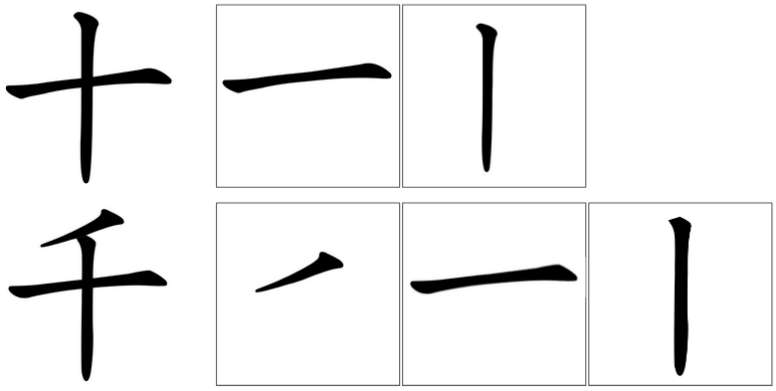


Figure 3.10: Examples from a Chinese Dataset where the first column *Left*: shows the input image and the remaining columns illustrate various stroke instances appearing in different channels.

Figure 3.11: Example of variations of strokes in a cat dataset. *Left*: The common way the cats are labeled. *Middle*: The two ears are labeled as one and the face as another. *Right*: The whole face is labeled as one.

region correctly. To leverage the strengths of both loss functions, they are combined in a weighted manner. The total loss is given as:

$$L = \beta * DL(y_T, y_p) + (1 - \beta) * S((y_T, y_p)) \tag{3.14}$$

where β is a hyperparameter, $DL(y_T, y_p)$ represents Dice Loss (see sub-section 3.3.2) and $S((y_T, y_p))$ represents the Margin-Regularised Loss Function (see sub-section 3.3.1). In our case, the value of β is 0.5 to equally leverage the benefits of both the loss functions. The value of β has been experimentally calculated (details in Section 4.2).

3.3.1 Margin-Regularized Loss Function

In binary classification, the margins are referred to as the distance between the data points and the hyperplane which separates each class. The aim is to find a hyperplane that maximizes this margin. Figure 3.12 illustrates the example of binary classification. Hinge Loss is one of the loss functions designed for maximum margin classification. It penalizes a correctly classified data point if its distance from the decision boundary is not large enough. Hinge Loss can be mathematically defined as:

$$h(y) = max(0, 1 - c.y(x_i)) \tag{3.15}$$

where $c.y(x_i)$ is the scalar product between true class labels (c= ± 1) and the raw output or score ($y(x_i)$) produced by the classifier for input x_i. If c and $y(x_i)$, both have the same sign, i.e., $y(x_i)$ is predicting correctly and $y(x_i) \geq 1$, the hinge loss will be 0. In cases where c and $y(x_i)$ both have same signs with $y(x_i) < 1$ i.e. $y(x_i)$ is predicting correctly but the margins are not enough or if c and $y(x_i)$, both have opposite signs i.e. the predictions are misclassified, the hinge loss varies linearly with respect to $y(x_i)$. These margins are used to scale the output of the model differently for each label. The Label-Distribution-Aware Margin (LDAM) loss function was introduced by Cao *et al*. [32] to handle heavy class imbalance by assigning larger margins to minority classes. The loss function achieves this by substituting the standard cross-entropy loss and incorporating re-weighting strategies to handle class imbalances. The re-weighting is performed by assigning different margins to each class based on their occurrences, calculated using the multi-class extension of hinge loss. The idea behind re-weighing is to make the loss function more adaptive by adjusting how different labels influence the feature space and loss calculation. These enforced margins are applied to the standard cross entropy to perform the smooth relaxation of the hinge loss.

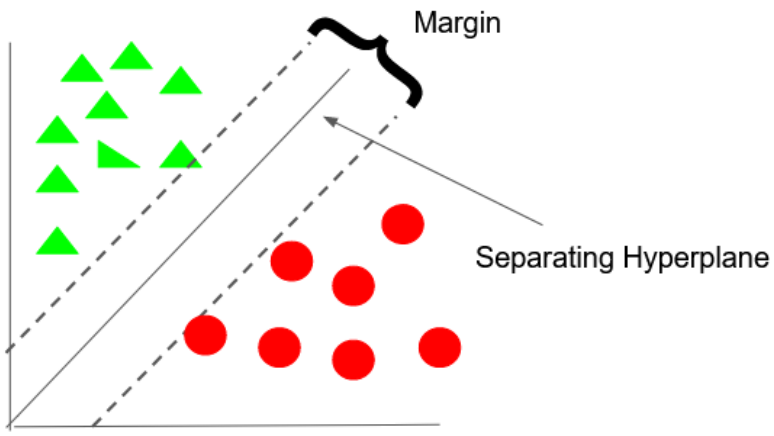


Figure 3.12: Example of binary classification, where data points from two classes, represented by red and green, are separated to achieve a maximum margin.

However, as stated in [32], this multi-label extension of the hinge loss suffers from optimization issues with more than 100 labels. This is because optimization algorithms, especially gradient-based methods, may struggle to efficiently navigate the non-differentiable and non-smooth properties of the hinge loss, leading to slower convergence and potential challenges in finding the optimal solution. Also, this loss function is implemented by constructing a matrix that consists of the number of occurrences of each class in a particular dataset and then regularizing the margins based on the majority and minority classes. However, since each stroke occurs only once for each image and can be found in any of the 128 channels, creating a matrix with 128 labels for millions of images is not feasible. Since the goal is to obtain wider margins that can be effectively used in margin scaling, with each logit intended to have significantly different values than the normalized logits obtained through softmax activation, we introduce a scaled sigmoid-like activation function to perform margin scaling for SCIS problems. Unlike softmax, which transforms logits into probabilities and ensures their sum is 1, a scaled sigmoid activation allows greater flexibility in creating distinctions between raw logits. This intentional deviation from the probability distribution function is designed to address specific objectives, such as achieving

wider margins, while considering potential challenges like vanishing gradient problems.

Inspired by the concept of Label-Distribution-Aware Margin (LDAM) [32], we developed a loss function that calculates the loss in three steps: (1) calculating the margins (2) margin scaling, and (3) adding a regularization term.

Let $\mathbf{y_T}$ and $\mathbf{y_p}$ represent the ground truth labels and predicted outputs. The margins are calculated in the following way (similar to a sigmoid function):

$$M(y_T, y_p) = \frac{1}{1 + se^{-y_T \cdot y_p}} \tag{3.16}$$

where s is a scaling factor. '.' represents the dot product between the ground truth labels and predicted outputs. The value of s, in our case, is 200 (this value has been determined experimentally).

Observations from Equation

Observation 1: (Asymptotic Behaviour) As $xy \to +\infty, M(x, y) \to 1$ and when $xy \to -\infty, M(x, y) \to 0$.

Proof: If $\lim_{xy \to +\infty} M(x, y)$, then $e^{-xy} \to 0$, as a result, s.$e^{-xy} \to 0$. Therefore, $(1 + se^{-xy}) \to 1$ and $M(x, y) \to 1$. Conversely, if $\lim_{xy \to -\infty} M(x, y)$, then $e^{-xy} \to \infty$, as a result, s.$e^{-xy} \to \infty$. Therefore, $(1 + se^{-xy}) \to \infty$ and $M(x, y) \to 0$.

Observation 2: Due to the randomness of the labels in the training data, the margin $M(x, y)$ is bounded between 0 and 1 for all the real values of x and y.

Proof: As we know, the value of e^{-xy} lies between 0 and 1, i.e., it will always be positive for all the real values of x and y. Therefore, the denominator $1 + se^{-xy}$ will always be positive and is bounded between 0 and $s + 1$. When the denominator is inversed, the bounded range is back to the interval (0, 1). As a result, $M(x, y)$ is always bounded by 0 and 1.

$$0 \leq M(x, y) \leq 1 \tag{3.17}$$

Equation 3.16 is used to scale margins and apply enforced margins in a cross-entropy loss function. The scaling of the margins is performed as follows:

$$\Delta_y = (1 - M(y_T, y_p)).y_T + M(y_T, y_p).y_T.y_p \tag{3.18}$$

where Δ_y represents the margin scaling term. The cross-entropy loss function with enforced margins is given below:

$$CE(y_T, y_p) = -\sum_{j=1}^{N} y_{T_i}. \log \Delta_y \tag{3.19}$$

In order to avoid overfitting and improve feature selection by encouraging a sparser set of feature weights, the L1 regularization term (i.e., the summation of the absolute values of the model's coefficient) is added to the above loss function. In our initial study, we tested our loss function for both L1 and L2 regularization terms and found that our training model converges better with L1 regularization than L2 regularization. As a result, we use the L1 regularization term for our loss function.

$$S(y_T, y_p) = CE(y_T, y_p) + \lambda \sum_{i}^{N} |y_{pi}| \tag{3.20}$$

where λ is a hyperparameter that represents the regularization strength to control the amount of regularization applied and y_p is the predicted model output.

3.3.2 Dice Loss

The Dice coefficient (DC) [26][25] is a widely used metric for segmentation tasks to measure the similarity between two images. The Dice coefficient measures how well the seg-

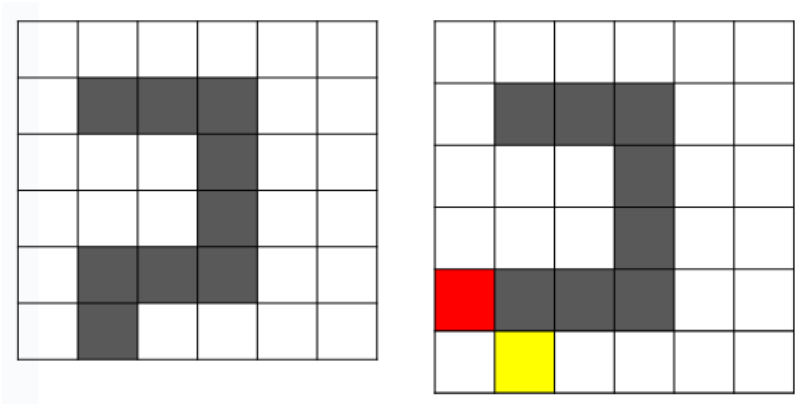


Figure 3.13: Example of Dice Loss. *Left*: ground truth and *Right*: predicted output. The grey boxes depict the overlap between the ground truth and the predicted segmentation output, representing True Positive (TP). The yellow and red boxes represent False Negative (FN) and False Positive (FP), respectively. The dice value for the above example, with 8 TP, 1 FP, and 1 FN, is given as 0.89.

mentation output overlaps with the ground truth. This measure ranges from 0 to 1, DC $\in [0, 1]$, where 1 represents a complete overlap, and 0 represents no overlap.

$$DC(y_T, y_p) = \frac{2y_T y_p}{y_T + y_p} = \frac{TP}{TP + FN} \tag{3.21}$$

The intersection term ($2y_T y_P$) in the Dice coefficient formula corresponds to the true positives (TP), while the union term ($y_T + y_P$) in the Dice loss formula encompasses the true positives (TP) and false negatives (FN). True positives (TP) are included in the union because they are part of both the ground truth and predicted masks, while false negatives (FN) are only part of the ground truth mask but not predicted. Figure 3.13 illustrates an example showing the overlap between the ground truth and the segmentation output. To minimize the dissimilarity between the predicted segmentation output and the ground truth, the Dice loss is mathematically expressed as:

$$DL(y_T, y_p) = 1 - DC(y_T, y_p) \tag{3.22}$$

Figure 3.14 illustrates a limitation of Dice loss. It can be clearly seen that Dice loss is

able to address the imbalance between the background and foreground, however, it ignores the difficulty of each example. These examples suffer from high variance, where a small difference in False Positive (FP) can lead to a significant change in the loss.

0	0	0	0	0	0
0	1	1	1	0	0
0	0	0	1	0	0
0	0	0	1	0	0
0	1	1	1	0	0
0	1	0	0	0	0

(a) Ground Truth

0.32	0.21	0.11	0.34	0.22	0.23
0.12	0.72	0.70	0.84	0.43	0.12
0.23	0.22	0.35	0.85	0.37	0.25
0.21	0.27	0.24	0.72	0.38	0.22
0.25	0.67	0.82	0.83	0.17	0.14
0.29	0.88	0.78	0.65	0.33	0.38

(b) Predicted Output 1

0.23	0.19	0.33	0.22	0.31	0.40
0.34	0.76	0.86	0.97	0.45	0.44
0.09	0.22	0.34	0.92	0.04	0.09
0.23	0.28	0.25	0.67	0.32	0.03
0.22	0.78	0.64	0.87	0.19	0.12
0.24	0.85	0.52	0.53	0.5	0.60

(c) Predicted Output 2

Figure 3.14: Example of limitation of Dice Loss. The grey and red boxes represent the TP and FP, respectively. The Dice loss calculated for both predicted probabilities with respect to the ground truth is 0.411 and 0.391. Despite having more FP in (c), the Dice loss is less than the one with less FP in (b).

Chapter 4

Results and Discussion

The performance of our proposed model named MFAU is assessed and compared against the state-of-the-art [17], as well as against the Attention UNet [3], a model which we have built upon. Section 4.1 provides information about the datasets. Section 4.2 discusses implementation details and provides a detailed explanation of the process involved in constructing our ground truth and vectorizing the output of our proposed approach. Section 4.3 analyzes the performance of our method in both quantitative and qualitative ways, evaluated for the six different datasets.

4.1 Datasets

Since our research objective is the vectorization of line art drawings, it is important to utilize datasets that include information about the stroke composition and stroke variations. By training our model on this type of datasets, we can easily extend our work to handle more complicated hand-drawn sketches, characters in various languages, mathematical formulas, and symbols. We evaluate our model on six datasets constructed by Kim *et al*. [17] for the vectorization of line art drawings. The six datasets belong to three semantically distinct categories: the Characters dataset (Chinese and Kanji), the Synthetic Random Lines

dataset, and the Sketch dataset (Baseball, Cat, and Multi-class). Figure 4.1 shows sample images from each dataset. Each of the six datasets consists of images in SVG format which are divided into training and testing sets constructed by Kim *et al*. [17]. In order to prevent overfitting and improve the model's ability to generalize on unseen datasets, we created a validation set by further splitting the training set into a 70:30 ratio. Table 4.1 shows the number of images present in each dataset for training, validation, and testing. Additionally, we are dealing with a specific use case that is inherently imbalanced in a heavy-tailed distribution manner. When evaluating the overall performance of each dataset, it is important to consider the distribution of the number of strokes within the dataset. Table 4.2 gives details about the number of vector paths (as the given images are in SVG file format, the strokes in SVG format are referred to as vector paths) used in the training/validation and testing datasets.

4.1.1 Characters Dataset

The characters dataset consists of vector graphics from Chinese and Kanji characters. The Chinese characters dataset comprises 9507 SVG format images. The training and testing set created by Kim *et al*. [17] consists of 8556 and 951 images, respectively. We further split the training set into 70:30 ratio; as a result, our validation set consists of 2567 images, and the training set now contains 5989 images. As shown in Table 4.2, the minimum and maximum number of vector paths present during the training/validation of Chinese characters are 1 and 33, respectively. The frequently occurring vector paths (also known as mode) during training/validation is 11. The mean ± standard deviation (which conveys the spread of our data) is 11.75 ± 4.39. During testing, the minimum, maximum and frequently occurring number of vector paths is 1, 28 and 10, respectively. The mean ± standard deviation for the testing dataset is 11.73 ± 4.32. Similarly, the Kanji characters dataset follows a split of 7216 training, 3093 validation, and 1145 testing data images.

The minimum, maximum and the mode of the number of vector paths present during the training of Kanji characters is 1, 30 and 12 and during testing is 1, 29 and 12, respectively. The mean ± standard deviation for the training and testing sets is 12.58 ± 4.79 and 12.39 ± 4.76, respectively. The characters dataset not only includes different characters but also varies according to the styles in which the user writes them.

4.1.2 Synthetic Random Lines Dataset

This dataset consists of a randomly generated combination of four lines and/or Bezier curves. This dataset consists of a total of 49,998 SVG format images. The training/validation and testing split (constructed in the same way as the Characters dataset) is shown in Table 4.1. Table 4.2 shows that the minimum, maximum, and average number of vector paths used during the training and testing phase is always four. This consistency arises from the fact that the dataset is constructed with four vector paths. Since there is no variability in the number of vector paths, the standard deviation for both the training and testing datasets is 0.

4.1.3 Sketches Dataset

This dataset consists of various sketches, drawn in numerous ways. This dataset has been constructed from the famous "Quick Draw" [1] dataset that contains 50 million drawings across 345 classes. Several players of The Quick Draw! game provided these drawings, showcasing different drawing styles. To maintain consistency between our method and the state-of-the-art method, we use the same classes as in [17]. Hence, we have the following classes: Baseball, Cat, and Multi-Class (a combination of the above two classes, in addition to Chandelier and Elephant). The Baseball, Cat and Multi-Class sketch datasets consist of a total of 135372, 123198 and 88888 SVG file formats, respectively. The training,

[1] https://github.com/googlecreativelab/quickdraw-dataset

	Chinese	Kanji	Random	Baseball	Cat	Multi-Class
Train	5989	7216	31499	85285	77616	56000
Val	2567	3093	13500	36551	33264	24000
Test	951	1145	4999	13536	12318	8888

Table 4.1: Number of images used for training, validation, and testing purposes.

Dataset	Minimum		Maximum		Mode		Mean		Std	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Chinese	1	1	33	28	11	10	11.75	11.73	4.39	4.32
Kanji	1	1	30	29	12	12	12.58	12.39	4.79	4.76
Random	4	4	4	4	4	4	4	4	0	0
Baseball	1	1	84	55	3	3	4.98	4.98	3.34	3.34
Cat	1	1	40	31	9	9	9.87	9.85	3.88	3.90
Multi-Class	1	1	115	48	3	3	7.03	7.0	4.50	4.45

Table 4.2: Minimum, maximum, mode, mean and standard deviation of the number of vector paths used in training/validation and testing phase for each dataset.

validation, and testing split is done in a similar fashion as for the Characters dataset and shown in Table 4.1. Table 4.2 gives details about the minimum, maximum, mode, mean and standard deviation of vector paths.

4.2 Implementation

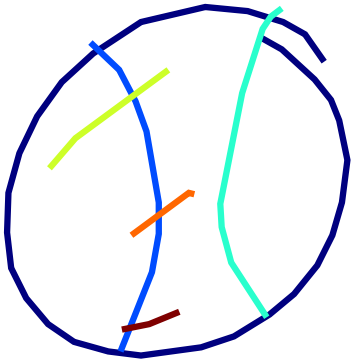
Our model is constructed using PyTorch. We conducted the training portion of our experiments on a GPU cluster containing four NVIDIA P100 GPU with 12 GB of memory. The inference was performed on a single desktop NVIDIA GeForce GTX 1660 Ti GPU with 6 GB of memory so that an accurate runtime measurement could be obtained.

The SVG files from [17] are stored with an image size of 64 x 64 for the Characters and Random lines datasets and an image size of 128 x 128 for the Sketch datasets. As a result, we use these respective image sizes. For the Characters and Random Lines datasets, we trained our model for 500 epochs with a batch size of 8. However, for the Sketch dataset,

(a) Chinese character dataset

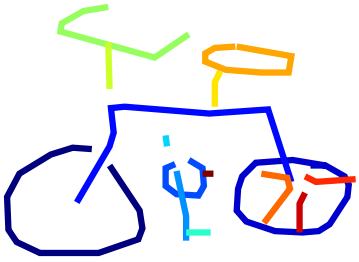
(b) Kanji character dataset

(c) Random Lines dataset



(d) Baseball sketch dataset

(e) Cat sketch dataset



(f) Multi-class sketch dataset

Figure 4.1: Sample images representing different datasets. *First Row:* Characters Dataset (Chinese and Kanji), *Middle Row:* Random Lines and Baseball Sketch Dataset, *Last Row:* Cat and Multi-Class Sketch Dataset. The colors are randomly assigned for presentation purposes, and the original input images are grayscale.

we used a batch size of 32 and trained for 200 epochs. The rationale behind using a higher batch size for the Sketch dataset is the significantly larger dataset size.

We trained our model using the Adam [39] optimizer and an initial learning rate of 1e-4. The maximum number of labels used for training our model is fixed at 128. Although this number is significantly greater than the highest number of vector paths among all the six datasets, the reason for choosing this large number was to maintain consistency [17], in which the authors also use a maximum of 128 labels in their model. Also, opting for this value did not have any negative impact on our model's performance and offered flexibility to accommodate datasets with a higher number of strokes.

4.2.1 Ground Truth Construction

The datasets from Kim *et al.* [17] are in SVG format. To use these data for training purposes, we need to convert them back to a raster format. The CairoSVG [2] library is used to convert SVG files to the PNG file format, outputting grayscale raster images that are used for training. Figure 4.2 shows the format in which SVG documents are stored. The SVG document is stored in an XML-type file that is used to describe the vector graphics. In order to create ground truth images, the SVG contents are accessed through the loop that iterates over each child path. Each child path is then converted to an XML string and further converted to the PNG format using the CairoSVG library. This implementation of converting vector images to raster images is similar to Kim *et al.* [17], however, our approach is implemented in PyTorch. We create a binary mask of each path extracted and then assign the same label to each pixel in that path. Finally, we combine all the labeled masks and create a final ground truth image. However, the vector paths are composed of control points instead of coordinates. Based on this information, it is challenging to determine which control points lead to the overlapping segments. When assigning labels

[2]https://pypi.org/project/CairoSVG/

```
<?xml version="1.0" encoding="utf-8" ?>
<svg baseProfile="tiny" height="128" version="1.2" viewBox="0,0,128,128" width="128"
xmlns="http://www.w3.org/2000/svg" xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xlink="http://www.w3.org/1999/xlink"><defs /><polyline fill="none"
points="82.4407,106.739 59.878,58.1424 56.8407,43.3898 56.4068,23.8644
58.5763,12.5831 60.7458,10.4136 67.6881,8.678 72.0271,8.678 74.6305,10.8475
76.3661,38.6169 82.0068,63.3492 85.478,86.7797 85.9119,111.078" stroke="#7cff79"
stroke-width="2" /><polyline fill="none" points="86.3458,109.7763 81.139,109.7763
78.9695,111.5119 75.4983,115.4169 74.6305,118.4542 82.8746,119.322
86.3458,117.1525 88.9492,112.3797 85.9119,109.3424" stroke="#7f0000"
stroke-width="2" /></svg>
```

Figure 4.2: *Left*: SVG image and *Right*: SVG file format that needs to be converted into PNG format so that it can be used as ground truth for our segmentation model. Each polyline (referred to as vector path in SVG file format) in this file represents a stroke of the line drawing.

to each pixel along the path, the overlapping region will ideally have a multi-assignment of labels, requiring additional efforts to handle this situation effectively. Therefore, we opt for a simpler labeling approach and assign each pixel in the overlap the label of the last path passing through it. This approach ensures a coherent labeling strategy while enabling us to still detect strokes with some minor discontinuities.

4.2.2 Potrace Algorithm

The output of our instance segmentation model represents a labeled version of the original input image. Each label extracted from the output of the segmentation model is isolated and transformed into a binary mask, where each mask corresponds to a distinct label. These binary masks (background as white and foreground as black) are passed through the Potrace software, designed primarily to trace black and white raster images and convert them into vectorized formats. Hence, the output of the Potrace software is a set of vectorized labels. Finally, all separated vectorized labels are combined, with careful consideration of their spatial positioning, to create a final vector image.

The Potrace algorithm, also known as polygon tracer, was introduced by Selinger *et al*. [5] to convert a black-and-white image from a raster to a vector outline, known as the tracing process. Polygons are used as an intermediate depiction of images, while the output of the algorithm is a smooth shape constructed using a Bezier curve. The authors explain the algorithm in four steps: ”In the first step, the bitmap is decomposed into a number of paths, which form the boundaries between black and white areas. In the second step, each path is approximated by an optimal polygon. In the third step, each polygon is transformed into a smooth outline. In an optional fourth step, the resulting curve is optimized by joining consecutive Bezier curve segments together where this is possible. Finally, the output is generated in the required format.”

In our scenario, each instance from the multi-channel output of the segmentation model is individually processed by the Potrace software. Figure 4.3 illustrates the Potrace process: First, the instance is converted into a binary format and passed through Potrace. The resulting SVG content is written in an empty file and assigned a color/hexadecimal code. Each subsequent instance is then passed through the potrace, and the vector path information is appended in the SVG file created along with the color/hexadecimal information for each instance.

4.3 Results and Analysis

4.3.1 Per-pixel IoU

To evaluate our results, we measure the per-pixel IoU between our final output and its corresponding ground truth. The IoU is calculated by dividing the number of pixels that belong to both the ground truth and the predicted segmentation output by the number of pixels that belong to either the ground truth or the predicted segmentation output. Figure 4.4 illustrates a simple example of the pixel-wise calculation of IoU for our instance segmentation

Figure 4.3: Potrace process for a three instance input. The binary mask of each instance is passed through Potrace and the vector path information is added to the SVG document.

0	0	0	0	0
0	1	1	2	0
0	0	0	2	0
0	0	0	2	0
0	3	3	3	0
0	0	0	0	0

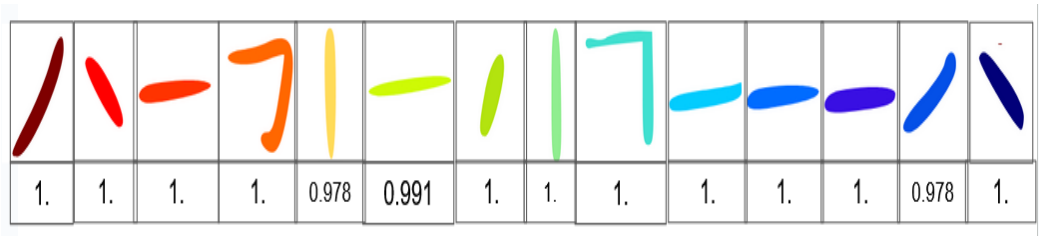
0	0	0	0	0
0	1	1	1	0
0	0	0	2	0
0	0	0	2	0
0	3	3	3	0
0	0	0	0	0

Figure 4.4: Example of pixel-wise IoU calculation. *Left*: shows the ground truth and *Right*: shows the predicted output. Label 2 is wrongly predicted as Label 1 in the predicted segmentation output while both Label 1 and Label 3 are predicted correctly. The IoU calculated is 1.0, 0.66, and 1.0 for each label, respectively. The overall IoU is calculated as 0.88.

model with multiple instances. In our use case, the order of the stroke matters, therefore, while comparing a correctly segmented stroke instance with the ground truth, if the labels are different (for example, if each pixel of a circle in the ground truth is labeled as 1 and in the predicted output the same circle is labeled as 2), the IoU for that particular stroke will be 0. Figure 4.5 shows pixel-wise IoU values calculated per instance, and the overall IoU is computed by averaging the IoU values calculated for each individual instance.

Our results are compiled and compared to reference methods in Table 4.3. As can be seen from Table 4.3, our method performed well in all datasets in comparison to Attention UNet. Overall, our method performed well in most of the datasets in terms of accuracy when compared to the state-of-the-art results. However, for the Chinese characters, Kim *et al*. [17] performed slightly better (+0.012) than our model. Our method requires a larger amount of data for increased performance; the Chinese dataset contains the least number of training images. However, we outperformed the other methods on the other datasets (excluding the Random Lines dataset). This points towards our model's ability to

(a) Ground Truth (b) Predicted Output (0.99)



(c) IoU Predictions per-instance (0.99 IoU)

Figure 4.5: *Top row* shows the ground truth and predicted output. *Bottom row* shows per instance IoU predictions. The labels are presented in order of their occurrence, and for each label, the corresponding prediction is displayed below it.

generalize effectively when provided with a large dataset.

The outlier in performance is the Random Lines dataset; our method outperformed [17] by an average of 0.0208 when excluding this dataset. The IoU for the Random Lines dataset for both Attention UNet and our method is less than 0.30. This dataset consists of only four labels, which is significantly fewer than other datasets that may contain up to 128 labels. Additionally, the size of the training dataset is larger than the Character dataset and less than the Sketch dataset. Despite having a relatively consistent data structure, our method did not perform well on the Random Lines dataset as compared to other datasets. This poor performance can be attributed to the totally random nature of the strokes, containing no spatial relationship between them, meaning they do not have any coherent patterns or structured configurations (such as, in the Cat Sketch dataset, in which a cat typically comprises a face, ears and whiskers). We hypothesize that as our model captures spatial

	Chinese	Kanji	Baseball	Cat	Multi-Class	Avg.	Random
Kim *et al.*	**0.958**	0.917	0.827	0.811	0.753	0.853	**0.872**
Attention UNet	0.884	0.918	0.853	0.433	0.742	0.766	0.276
Ours	0.937	**0.946**	**0.869**	**0.831**	**0.786**	**0.874**	0.271

Table 4.3: Per-pixel intersection-over-union (IoU) performed on the test set. Best results are shown in bold font. The IoU average excludes the Random Lines dataset, which was included as a counter-example.

relationships well, and in the absence of these relationships, we are essentially training on "noise". Thus, this serves as an illustrative example, showcasing our model's capacity to leverage spatial relationships between instances. It's important to note that these insights are post-analysis, offering a retrospective understanding. Moreover, our model consistently performs effectively across datasets with more semantically structured contents.

Selection of Scaling Factor s

The scaled sigmoid function uses s as a scaling factor for calculating margins used in the Margin-Regularised Loss Function (MRLF). To choose the correct value of s, we carried out several experiments with different values of s. All the experiments are carried out on the Multi-Class dataset as this dataset is the most complicated among all the six benchmark datasets with large variations in the number of instances, and it also has some overlap with the Baseball and Cat drawings. Table 4.4 displays different IoU values achieved by varying s. We start by setting s to 0.1, yielding an IoU of 0.70. We then increase the value of s to 10, 30, 100, 200, and 300. An increasing trend in IoU can be seen by increasing the value of s up to 200, with values of 0.727, 0.731, 0.774, 0.786 and 0.778. Since the accuracy sees a decreasing trend at $s = 300$, we choose 200 as the scaling factor for all the datasets.

Selection of Hyperparameter β in the compounded loss function

As discussed in Section 3.3, our loss function is a weighted combination of DL and MRLF. To find the optimal value of the hyperparameter β in the compounded loss function, we

s	IoU ↑
0.1	0.70
10	0.727
30	0.731
100	0.774
200	**0.786**
300	0.778

Table 4.4: Experiments with different values of s and the corresponding IoU values as the output. The up arrow indicates that the value should be high.

DL	MRLF	IoU ↑
0.5	0.5	**0.786**
0.3	0.7	0.678
0.7	0.3	0.713

Table 4.5: Table dislays the variation in the value of β for DL and MRLF. The last column depicts the corresponding IoU values. The up arrow indicates that the value should be high.

carried out experiments by varying the value of β for the Multi-Class dataset. Table 4.5 shows different values of β for DL and MRLF and their corresponding IoU values. We start by setting the value of β to 0.5 for both Dice Loss and MRLF in order to give them an equal contribution, yielding an IoU of 0.786. We then changed the value of β to 0.3 for Dice Loss and 0.7 for MRLF, this time yielding an IoU of 0.678. When changing the value of β to 0.7 for Dice Loss and 0.3 for MRLF, the IoU is 0.713. The value of β has a significant impact on the IoU; we select 0.5 for the main experiments.

4.3.2 Computation Time

The average execution time per image is shown in Table 4.6. We have compared the average computation time taken per image at the inference of our proposed segmentation model with Kim *et al*. 's method and Attention UNet. However, to ensure a fair comparison with their results, the table excludes the execution time required for vectorization using Potrace, as the state-of-the-art method focused only on assessing their model's performance instead of the time taken by an iterative software.

Comparisons with State-of-the-Art method [17]

As can be concluded from Table 4.6, our approach takes less than 50 milliseconds and is 1711500% faster than the state-of-the-art method [17] across all the datasets. Our approach is 6200 and 2800 times faster than Kim *et al*. [17] for the Characters datasets (Chinese and Kanji, respectively). For the Random Lines dataset, our approach is 776 times faster, while in the case of the Sketch datasets, our approach is 17800, 48000, and 10400 times faster than the state-of-the-art for Multi-Class, Baseball, and Cat drawings, respectively. The Character datasets take less execution time than all the other datasets. This can be attributed to their smaller image size of 64x64, and the maximum number of instances being just 30 and 33. In contrast, the size of the images in the Sketch dataset is 128x128, and considerable variability in the number of instances ranging from 1 to 115 makes the model more complex. As a result, the computation time is comparatively higher for the Sketch datasets as compared to the Character datasets. However, despite having only four instances, the Random Lines dataset takes more time than the Characters datasets of the same image size but a higher number of instances. This anomaly can be ascribed to the model's inability to generalize effectively on the Random Lines dataset (explained in Section 4.3.1). Considering the Kanji Characters dataset, the average execution time taken for images containing less than 12 instances (this number represents the frequently occurring number of instances in this particular dataset) is 0.0093. In contrast, for images containing more than 12 instances, the average execution time is 0.0096. Images featuring exactly 12 exhibit an average execution time of 0.0095. For the Sketch datasets such as Baseball, the average execution time for images containing less than 3 (mode of the Baseball dataset), greater than 3, and exactly 3 instances is 0.01795, 0.01800, and 0.01798.

Comparisons with Attention UNet [3]

While comparing our method with Attention UNet in terms of execution time, it can be seen our approach is 64% faster across all datasets. This improvement in execution time is due to the replacement of skip connections that concatenate information from the encoding and decoding layer at each stage, with Highway Skip Connections that pass only selective information at each stage.

	Chinese	Kanji	Random	Multi-Class	Baseball	Cat
Kim *et al*.	54.9	26.589	8.46	409	853	233
Attention UNet	0.0106	0.0154	0.0146	0.0389	0.0404	0.0365
Ours	**0.0089**	**0.0095**	**0.0109**	**0.0231**	**0.0179**	**0.0223**

Table 4.6: Average execution time taken by a single image at inference (in seconds).

4.3.3 Computation Complexity

GFLOPs (Giga-Floating Point Operations Per Second) is a unit of measure used to calculate the computational complexity of a model. It estimates the number of arithmetic operations involving floating-point numbers that the model performs in one second and expressed as giga per second. The GFLOPs are calculated by dividing the total number of floating point operations by the elapsed time in seconds. The result is further divided by 10^9 to represent the units in giga.

$$FLOPs = \frac{N}{t} \tag{4.1}$$

$$GFLOPs = \frac{FLOPs}{10^9} \tag{4.2}$$

where N represents the number of floating point operations and t represents the elapsed time in seconds. Higher values of GFLOPs suggest more difficult computational tasks, whereas lower values imply lighter and faster models. However, it is important to keep in mind

that GFLOPs cannot entirely capture how fast/light a model can be. Despite theoretically illustrating the highest level of performance a model could achieve in an ideal situation, they tend to ignore several important factors that could account for the computational performance, like memory access, parallelization, and other system constraints. Hence, they cannot be treated criterion of comparison for a model's performance. The computation complexity in terms of GFLOPs and the number of parameters of our model and compared approaches are shown in Table 4.7.

Comparisons with State-of-the-Art method [17]

Despite being less complex and having fewer parameters, the state-of-the-art model [17] requires a significantly higher execution time. This method is trained on two separate neural networks, named PathNet and OverlapNet. Both networks are made up of 20 sequential blocks, with each containing 64 3x3 filters. The only difference between these two networks is their inputs and the use of different non-linear activation functions. The predictions from these models are combined by solving the Markov Random Field (MRF), which is a crucial and time-consuming step. Since the iterative optimization process mostly reuse the already created parameters and GFLOPs calculate the floating point operation in a model (iterative optimization is not part of a model), the calculation of the number of model parameters and GFLOPs does not include the iterative optimization of the MRF. As a result, the parameters and the floating point operations calculated are significantly less than ours. Hence, comparing our model with the state-of-the-art method based on the computational complexity of the model may not be relevant.

Comparisons with Attention UNet [3]

In Section 3.1, we have provided a theoretical explanation of the computational benefits of replacing skip connections with Highway Connections in our approach compared to

Model	Number of Parameters ↓	GFLOPs ↓
Kim *et al.*	1.34 M	0.00266
Attention UNet (Vanilla)	34.9 M	4.19
Ours	32.8 M	3.79

Table 4.7: Computational Complexity, measured in terms of number of parameters and GFLOPS. The down arrow indicates that it is better to have low values.

Attention UNet. Experimentally, it can be seen from Table 4.7 that our proposed model is better than the Attention UNet method in terms of both model parameters and model complexity. Attention UNet has more than 2.1 million additional parameters compared to our proposed approach. Additionally, in terms of GFLOPs, our model achieves a reduction of 0.4 compared to Attention UNet. This highlights our model's efficiency in terms of both parameter count and computational complexity.

4.3.4 Qualitative Analysis

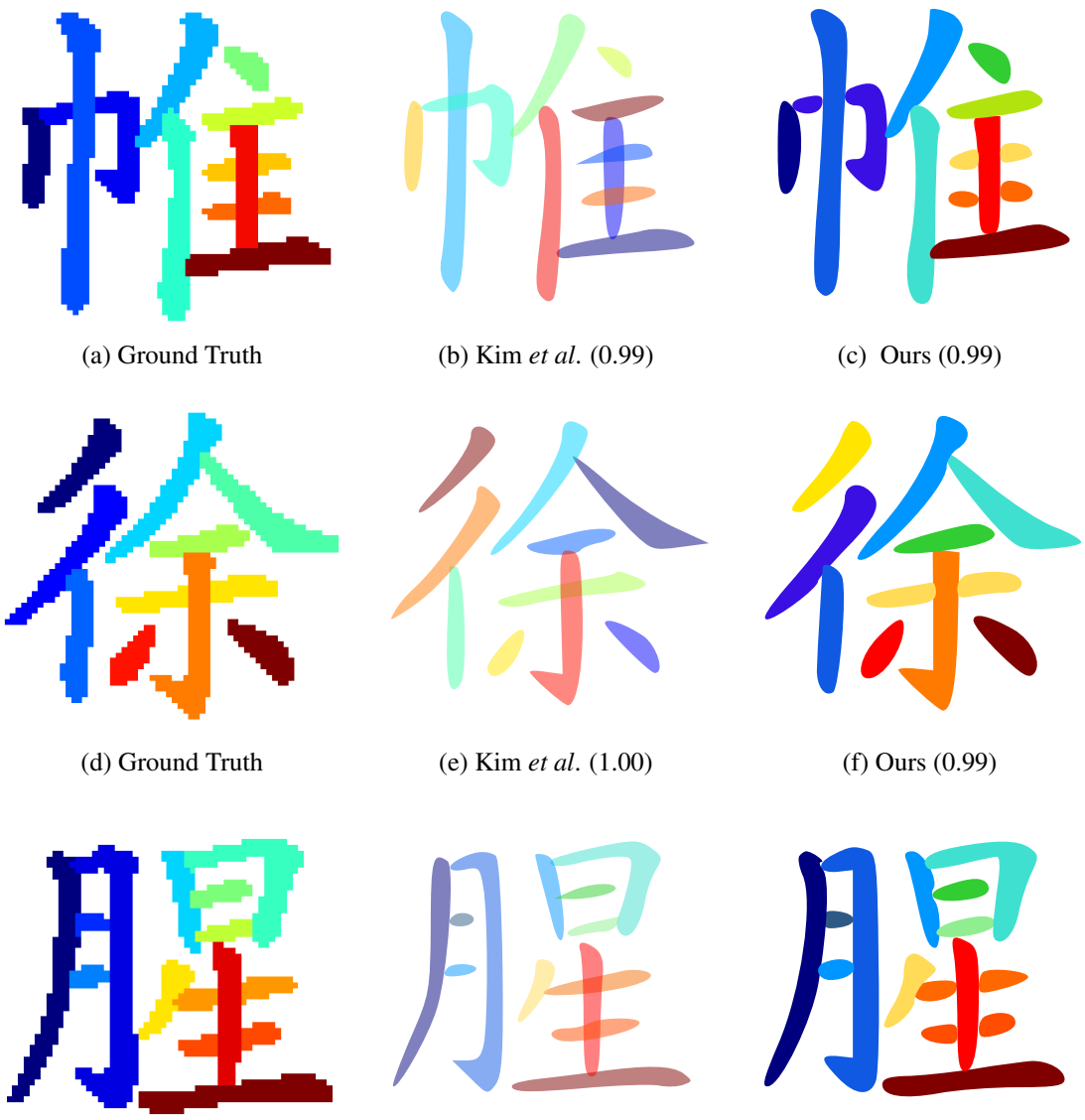
We visually compare our outputs with those of [17] using examples from all datasets except the random lines. Although both methods are able to segment correct instances of strokes in most cases, both have distinct shortcomings.

Figure 4.6, Figure 4.7, and Figure 4.8 present examples where our method performed qualitatively better than [17]. As shown in Figure 4.6 and Figure 4.7, despite the presence of a missing pixel in the overlapping regions of the ground truth, we are still able to maintain continuity in the strokes. It can be clearly seen in Figure 4.6c that the red stroke overlaps with yellow and orange strokes. Even with a missing pixel in the overlapping regions of the ground truth (as seen in the yellow and orange strokes), we can still preserve the continuity of the strokes, i.e., the model is able to accurately predict the next pixel after the missing pixel within the same vector path of the stroke in yellow and orange. This implies that our model is able to follow the perceptual grouping principle of continuity. However, further research is needed to fully explore the implications of perceptual grouping in this context.

As discussed in Section 3.3, our model is able to handle various drawing styles. This can be attributed to the incorporation of margin scaling in our loss function. The margin scaling is applied on the margins calculated by using a scaled sigmoid function. This scaling function plays a crucial role in ensuring an effective separation between labels, making them easily distinguishable, thus, allowing the model to handle the variations present in the stroke instances. A user can draw a cat in various styles, such as using a single stroke for the whole face, two separate strokes for the ears and face, or individual strokes for the ears and face. In Figure 4.8c, our model not only segments the instances but also infers the user's drawing style. This stands in contrast to [17], as shown in Figure 4.8b, which erroneously predicts the face and ears as one stroke (i.e. the most common drawing style). Figure 4.8e shows that B. Kim *et al.* method predicts the circle of the baseball as two separate strokes, after an overlap occurs between the navy blue stroke and the purple stroke. However, the outcome of our model shown in Figure 4.8f is able to predict the circle as a single instance correctly. As seen in Figure 4.8i, our model correctly identifies and segments the closed red curve as a single, closed entity. Unlike the B. Kim *et al.* [17] method shown in Figure 4.8h , where the closed red curve is segmented into two separate strokes. This implies our model can incorporate the perceptual grouping principle of closure.

Figure 4.9 and Figure 4.10 shows some shortcomings of our method when compared to [17]. In the character dataset (shown in Figure 4.9c and Figure 4.9f), it can be observed that multiple instances are assigned to a single stroke instance, resulting in a phenomenon known as "vector soup". However, if human intervention is allowed as a postprocessing step, the number of required edits would be minimal; thus our output could be successfully utilized for further editing. For example, Figure 4.9f can be corrected with only two edits by correcting vector paths for the purple stroke ending with an orange label at the end and correcting an aqua label present on the orange vector path. In Figure 4.10c and Figure 4.10f, some of the instances were mislabeled, however, our model accurately identifies circles in

both categories, unlike [17]. Moreover, our model correctly identifies the instances of ears and eyes of the cat along with the circle. We hypothesize that in most cases, the circle is present in the first channel of these two class prototypes. As a result, our model is able to predict similar patterns in a specific class prototype. This shows the ability of the model to follow the perceptual grouping principle of similarity. Overall, it can be concluded that our model not only is able to infer the drawing styles of various users but also potentially adheres to three main perceptual grouping principles: similarity, continuity, and closure.



(a) Ground Truth (b) Kim *et al.* (0.99) (c) Ours (0.99)

(d) Ground Truth (e) Kim *et al.* (1.00) (f) Ours (0.99)

(g) Ground Truth (h) Kim *et al.* (0.97) (i) Ours (0.99)

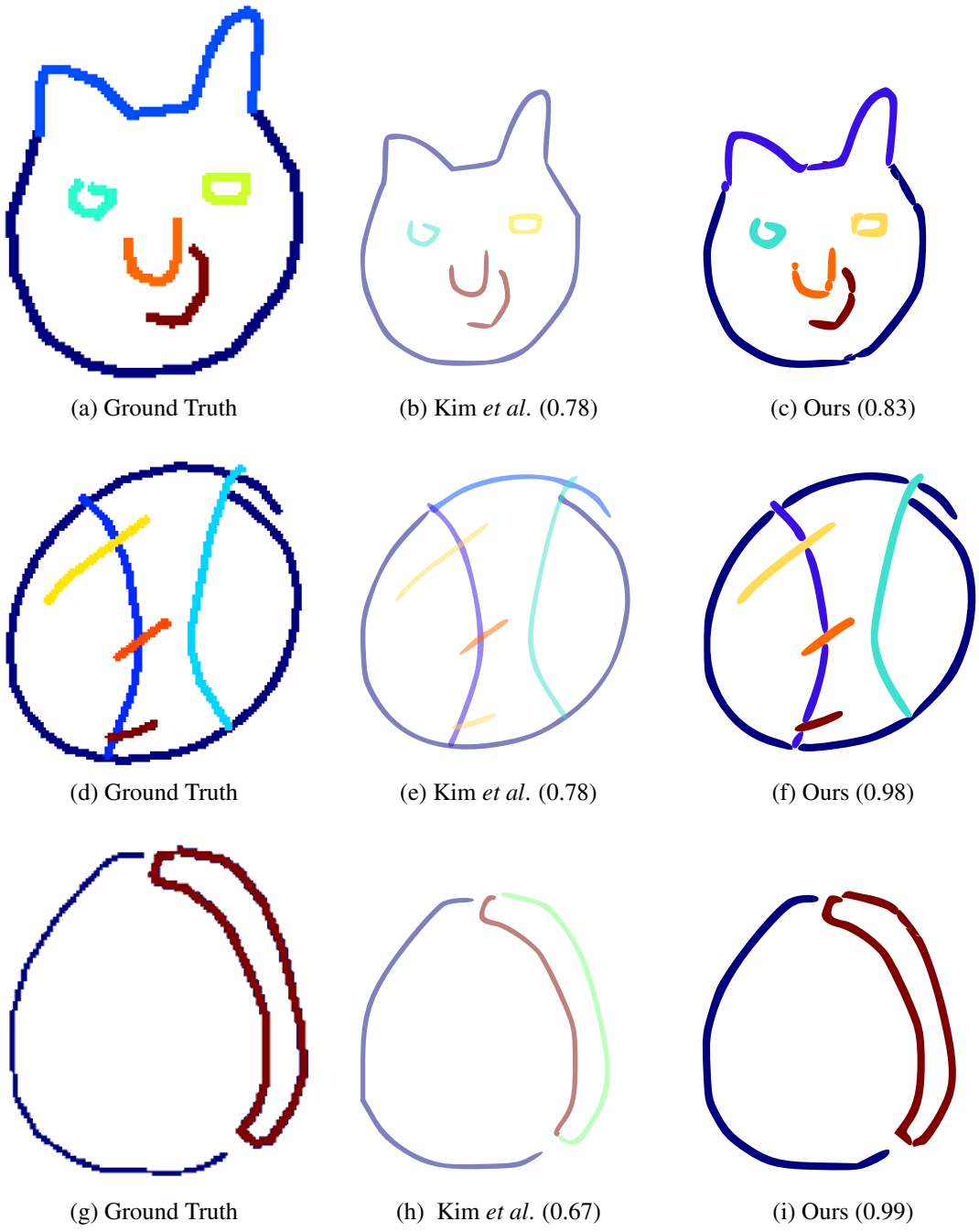
Figure 4.6: Stroke-based comparison between Kim *et al.* and our method on the Chinese Characters dataset. The ground truth is shown on the left for each case, the results of Kim *et al.* in the middle, and the outcome of our model is shown on the right. The per-image IoU is also shown for a fair comparison between the two models.

(a) Ground Truth (b) Kim *et al.* (0.99) (c) Ours (0.99)

(d) Ground Truth (e) Kim *et al.* (0.96) (f) Ours (0.99)

(g) Ground Truth (h) Kim *et al.* (0.94) (i) Ours (0.99)

Figure 4.7: Stroke-based comparison between Kim *et al.* and our method on the Kanji Characters dataset. The ground truth is shown on the left for each case, the results of Kim *et al.* in the middle, and the outcome of our model is shown on the right. The per-image IoU is also shown for a fair comparison between the two models.



(a) Ground Truth (b) Kim *et al.* (0.78) (c) Ours (0.83)

(d) Ground Truth (e) Kim *et al.* (0.78) (f) Ours (0.98)

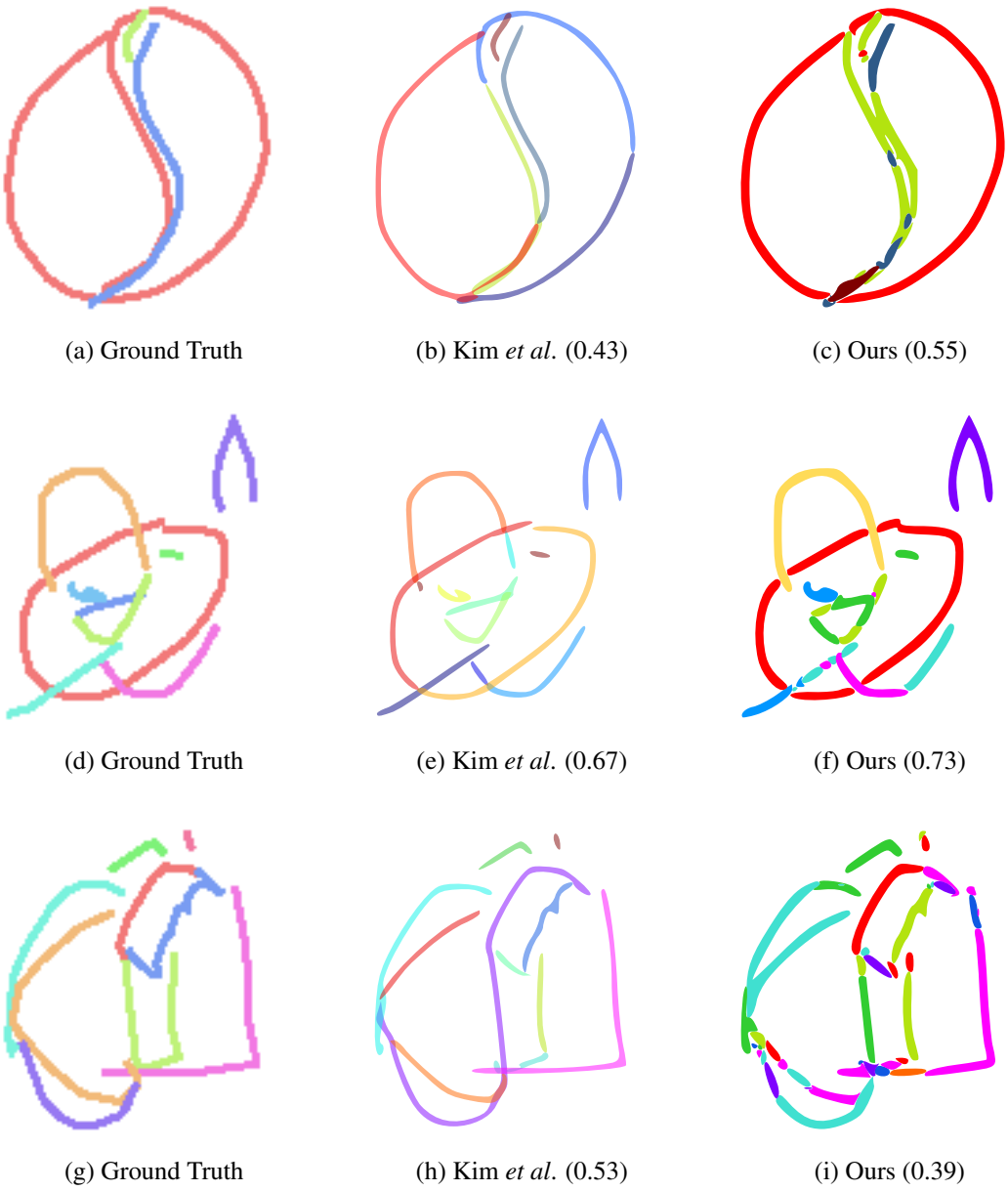
(g) Ground Truth (h) Kim *et al.* (0.67) (i) Ours (0.99)

Figure 4.8: Stroke-based comparison between Kim *et al.* and our method on the Sketches dataset, where *First Row:* represents Cat dataset, *Middle Row:* Baseball dataset and *Last Row:* represents Backpack class from Multi-Class dataset. This figure shows some of the cases where Kim *et al.* 's method did not perform well in comparison to our method. The per-image IoU is also shown for a fair comparison between the two models.

(a) Ground Truth (b) Kim *et al.* (0.91) (c) Ours (0.83)

(d) Ground Truth (e) Kim *et al.* (1) (f) Ours (0.92)

Figure 4.9: Stroke-based comparison between Kim *et al.* and our method on Characters dataset, where *First Row:* represents Chinese Characters, and *Last Row:* represents Kanji Characters. This figure shows some of the cases where our method did not perform well on the Characters dataset. The per-image IoU is also shown for a fair comparison between the two models.



(a) Ground Truth (b) Kim *et al.* (0.43) (c) Ours (0.55)

(d) Ground Truth (e) Kim *et al.* (0.67) (f) Ours (0.73)

(g) Ground Truth (h) Kim *et al.* (0.53) (i) Ours (0.39)

Figure 4.10: Stroke-based Comparison between Kim *et al.* and our method on Sketches dataset, where *First Row:* represents Cat dataset, *Middle Row:* Baseball dataset and *Last Row:* represents Backpack class from Multi-Class dataset. This figure shows some of the cases where our method did not perform well on the Sketch dataset. The per-image IoU is also shown for a fair comparison between the two models.

Chapter 5

Conclusions

The research in this thesis was motivated by the importance of vector graphics and the rationale behind converting raster images to vectors, specifically for line drawings. The state-of-the-art paper [17] presented an effective yet time-consuming solution, highlighting the need for a more efficient alternative.

We proposed a novel segmentation method to perform Single-Class Instance Segmentation (SCIS) that processes the input raster image by labeling each pixel as belonging to a particular stroke instance. Our novel architecture, named Multi-Focus Attention UNet (MFAU), builds upon Attention UNet by introducing Multi-Focus Attention Gates (MFAG) and Highway Skip Connections. MFAG are added in place of Attention Gates to improve the higher-level feature representations by capturing the global and local context of the image. Highway skip connections enhance the computational efficiency of the model by incorporating adapting gating units that learn to select which information needs to be passed or transformed. Our loss function includes a margin-regularised component which allows us to handle successfully a heavy-tailed label distribution, as well as infer correctly the user's drawing style.

We assessed our approach by comparing our results on six datasets (i.e. Chinese, Kanji,

Random Lines, Baseball, Cat, and Multi-Class) against a state-of-the-art method (i.e., Kim *et al.*) and a reference method (i.e., Attention UNet). Our approach is significantly faster, exceeding the state-of-the-art method by seven orders of magnitude and outperforming by 0.0208 per-pixel IoU (excluding the Random Lines dataset). Compared to the reference method, our approach demonstrates a 64% increase in speed and achieves a higher per-pixel IoU by 0.103, excluding the Random Lines dataset. Our counter-example, the Random Lines dataset, demonstrated our model's ability to leverage the spatial relationship between each stroke instance. The stroke-based qualitative comparison implies that our model has the potential to follow the three main perceptual grouping principles: similarity, continuity, and closure.

As observed in the qualitative analysis, there are certain examples where multiple labels are being assigned to a single stroke, resulting in a phenomenon known as "vector soup". This phenomenon can be easily mitigated by utilizing some nearest-neighbor search algorithms, which assign the same label to an incorrect pixel as their neighbors. However, most of these post-processing steps available to us are time-consuming. As a result, there is a trade-off between accuracy and processing time. In the future, we aim to explore some faster post-processing techniques with a more balanced trade-off. Additionally, our current vectorization model produces pixel-based segmentation results, making the pixels susceptible to misclassification and contributing to the multi-assignment of labels within a single stroke. Therefore, our future work aims to extend the vectorization model from pixel-based to a unified stroke-based segmentation. This segmentation would assign a single label to an entire stroke rather than assigning individual labels to each pixel in a stroke. This would reduce the likelihood of misclassification of pixels in a stroke, as in this case the stroke would be considered a single entity during the segmentation process.

While our current research focuses on line drawings, our future objectives involve extending our work to complex line art drawings, particularly those that would significantly

benefit from image vectorization when shown on large displays. These complex designs may include architectural blueprints, elaborate hand-drawn patterns, circuit designs, complex engineering schematics, abstract art, scientific diagrams, calligraphy, and more. These drawings feature complicated details that leverage the advantages of scalability and precision offered by the image vectorization process, especially when displayed on large screens. Moreover, by incorporating these designs we can easily integrate our model into an interactive software, providing a more sophisticated way to editability with vectorized output. Additionally, we plan to extend our work to three-dimensional scenes by transitioning from pixel-based representation to voxel-based representation by adding a third dimension to our segmentation model, allowing for the vectorization of 3D models. This extension could find applications in areas like CAD or virtual reality.

Bibliography

[1] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[3] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention u-net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018.

[4] P de Casteljau. "courbes et surfaces à poles," technical report. *Citroen, Paris*, 1963.

[5] Peter Selinger. Potrace: a polygon-based tracing algorithm, 2003.

[6] Javier Jimenez and Jose L Navalon. Some experiments in image vectorization. *IBM Journal of research and Development*, 26(6):724–734, 1982.

[7] Pekka Kultanen. Randomized hough transform (rht) in engineering drawing vectorization system. *Proc. MVA*, pages 173–176, 1990.

[8] Markus Roosli and Gladys Monagan. Adding geometric constraints to the vectorization of line drawings. In *Graphics Recognition Methods and Applications: First International Workshop University Park, PA, USA, August 10–11, 1995 Selected Papers 1*, pages 49–56. Springer, 1996.

[9] Dov Dori. Orthogonal zig-zag: an algorithm for vectorizing engineering drawings compared with hough transform. *Advances in Engineering Software*, 28(1):11–24, 1997.

[10] Karl Tombre, Christian Ah-Soon, Philippe Dosch, Gérald Masini, and Salvatore Tabbone. Stable and robust vectorization: How to make the right choices. In *Graphics Recognition Recent Advances: Third International Workshop, GREC'99 Jaipur, India, September 26–27, 1999 Selected Papers 3*, pages 3–18. Springer, 2000.

[11] Alexandra Bartolo, Kenneth P Camilleri, Simon G Fabri, Jonathan C Borg, and Philip J Farrugia. Scribbles to vectors: preparation of scribble drawings for cad interpretation. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, pages 123–130, 2007.

[12] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. Fidelity vs. simplicity: a global approach to line drawing vectorization. *ACM Transactions on Graphics (TOG)*, 35(4):1–10, 2016.

[13] Patryk Najgebauer and Rafał Scherer. Inertia-based fast vectorization of line drawings. In *Computer Graphics Forum*, volume 38, pages 203–213. Wiley Online Library, 2019.

[14] Brian Wood. *Adobe illustrator CS6 classroom in a book*. Adobe Press, 2012.

[15] Gary David Bouton. *CorelDRAW X7*. Mcgraw-hill Education-Europe, 2014.

[16] Naoto Inoue and Toshihiko Yamasaki. Fast instance segmentation for line drawing vectorization. In *2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM)*, pages 262–265. IEEE, 2019.

[17] Byungsoo Kim, Oliver Wang, A Cengiz Öztireli, and Markus Gross. Semantic segmentation for line drawing vectorization using neural networks. In *Computer Graphics Forum*, volume 37, pages 329–338. Wiley Online Library, 2018.

[18] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE ICCV*, Oct 2017.

[19] Ma Yi-de, Liu Qing, and Qian Zhi-Bai. Automated image segmentation using improved pcnn model based on cross-entropy. In *Proceedings of 2004 International Symposium on Intelligent Multimedia, Video and Speech Processing, 2004.*, pages 743–746. IEEE, 2004.

[20] Vasyl Pihur, Susmita Datta, and Somnath Datta. Weighted rank aggregation of cluster validation measures: a monte carlo cross-entropy approach. *Bioinformatics*, 23(13):1607–1615, 2007.

[21] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *Proceedings of the IEEE international conference on computer vision*, pages 1395–1403, 2015.

[22] BN Chaithanya, TJ Swasthika Jain, A Usha Ruby, and A Parveen. An approach to categorize chest x-ray images using sparse categorical cross entropy. *Indonesian Journal of Electrical Engineering and Computer Science*, pages 1700–1710, 2021.

[23] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[24] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[25] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*, pages 240–248. Springer, 2017.

[26] Th A Sorensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on danish commons. *Biol. Skar.*, 5:1–34, 1948.

[27] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. Tversky loss function for image segmentation using 3d fully convolutional deep networks. In *International workshop on machine learning in medical imaging*, pages 379–387. Springer, 2017.

[28] Maxim Berman, Amal Rannen Triki, and Matthew B Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4413–4421, 2018.

[29] Claudio Gentile and Manfred KK Warmuth. Linear hinge loss and average margin. *Advances in neural information processing systems*, 11, 1998.

[30] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[31] Saeid Asgari Taghanaki, Yefeng Zheng, S Kevin Zhou, Bogdan Georgescu, Puneet Sharma, Daguang Xu, Dorin Comaniciu, and Ghassan Hamarneh. Combo loss: Handling input and output imbalance in multi-organ segmentation. *Computerized Medical Imaging and Graphics*, 75:24–33, 2019.

[32] Kaidi Cao, Colin Wei, Adrien Gaidon, Nikos Arechiga, and Tengyu Ma. Learning imbalanced datasets with label-distribution-aware margin loss. *Advances in neural information processing systems*, 32, 2019.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[34] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[35] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. *Advances in neural information processing systems*, 28, 2015.

[36] Meng-Hao Guo, Tian-Xing Xu, Jiang-Jiang Liu, Zheng-Ning Liu, Peng-Tao Jiang, Tai-Jiang Mu, Song-Hai Zhang, Ralph R Martin, Ming-Ming Cheng, and Shi-Min Hu. Attention mechanisms in computer vision: A survey. *Computational visual media*, 8(3):331–368, 2022.

[37] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9157–9166, 2019.

[38] Alex Graves and Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012.

[39] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.