UNIVERSIDADE D
COIMBRA

Francisco José Brilhante Fernandes

Generative Modeling for Automated Type Design

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, supervised by Prof. João Nuno Gonçalves Costa Cavaleiro Correia and Daniel Filipe Santos Lopes and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2023

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Francisco José Brilhante Fernandes

Generative Modeling for Automated Type Design

Dissertation in the context of the Master in Informatics Engineering, specialization in Intelligent Systems, supervised by Prof. João Nuno Gonçalves Costa Cavaleiro Correia and Daniel Filipe Santos Lopes and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2023

DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA
FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Francisco José Brilhante Fernandes

Modelos Generativos para Automatização de Design Tipográfico

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Sistemas Inteligentes, orientada pelo Professor Doutor João Nuno Gonçalves Costa Cavaleiro Correia e Daniel Filipe Santos Lopes e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho 2023

Acknowledgements

I would like to express my gratitude to all the individuals who directly or indirectly contributed to the journey I have embarked on this year, which now culminates in the completion of this dissertation. Without your support and messages of confidence, it would not have been possible to reach this point.

I am deeply grateful to my advisors, Prof. João Nuno Gonçalves Costa Cavaleiro Correia and Daniel Filipe Santos Lopes, for their invaluable knowledge and wise guidance. Their vision, expertise, and unwavering support have played a crucial role in transforming this project into a reality. Additionally, I extend my thanks to Prof. Catarina Silva and Prof. Bernardete Ribeiro for introducing me to the world of scientific research and for instilling in me a belief in my own capabilities.

I would also like to express my appreciation to the Centre for Informatics and Systems of the University of Coimbra for welcoming me to a conducive working environment and also providing the necessary infrastructure to facilitate the completion of this work.

Last but certainly not least, I extend my heartfelt thanks to my friends and family. I am grateful to my family for teaching me the immeasurable value of perseverance required to climb all the vital steps and for instilling confidence in the decisions I have made throughout this past year.

Francisco José Brilhante Fernandes

Abstract

Typography design is often a time-consuming process that requires skill, expertise, and experience. To hasten this process, machine learning techniques have been explored to automatically build entire typographical fonts based on a few manually designed glyph samples. However, such existing systems are still not practical to aid the creation of functional fonts as their outputs are either in raster formats or too noisy, therefore requiring considerable manual post-edition. In this dissertation, we explore a combined approach using deep generative techniques such as Generative Adversarial Networks and Diffusion Models to generate fonts in scalable vector graphic formats, allowing for greater flexibility and control in the design process. Moreover, we put forth innovative contributions to existing generative systems, yielding significant quantitative improvements. We propose a new attention-based loss function that aims to better guide the learning process of font generation models. Such improvements are followed by the study of optimal input selection in multi-shot scenarios. We also devise one of the first diffusion networks specialized in glyph generation, which successfully tackles much of the visual defects inherent to previous raster methods. These contributions are embedded in a web application that allows users to generate complete fonts that closely match the style and features of the provided glyph samples, offering a tool for type designers and practitioners to create custom typefaces.

Keywords

Typography, Fonts, Automatic Glyph Generation, Generative Models, Generative Adversarial Networks, Autoencoder, Style Transfer, Diffusion Model

Resumo

O desenho tipográfico é frequentemente um processo moroso que requer competência, perícia e experiência. Para acelerar este processo foram já exploradas técnicas de aprendizagem de máquina para gerar fontes tipográficas completas baseadas num conjunto pequeno de amostras de glifos desenhados manualmente. Contudo, tais sistemas ainda não são práticos para ajudar à criação de fontes funcionais, uma vez que os seus resultados ou estão em formatos rasterizados, ou contêm demasiado ruído, exigindo por isso um grande esforço de pós-edição manual. Nesta dissertação, exploramos uma abordagem combinatória que utiliza técnicas generativas profundas, como Redes Adversárias Generativas e Modelos de Difusão para gerar fontes em formatos vetoriais, permitindo uma maior flexibilidade e controlo no processo de design. Para além disso, contribuímos com modificações inovadoras a sistemas generativos já existentes, produzindo melhorias quantitativas significativas. Estas modificações incluem novas funções de perda baseadas em atenção, que visam orientar melhor o processo de aprendizagem de modelos de geração de fontes. Estas melhorias são seguidas de um estudo da seleção de exemplos iniciais ótimos em cenários de múltiplas entradas. Também criámos uma das primeiras redes de difusão especializadas na geração de glifos, que resolve com sucesso muitos dos defeitos visuais inerentes aos métodos anteriores de rasterização. Estes contributos estão englobados numa aplicação *online* que permite aos utilizadores gerar tipos de letra completos que se aproximam do estilo e das características das amostras de glifos fornecidas, oferecendo uma ferramenta para que designers e profissionais do sector possam criar tipos de letra personalizados.

Palavras-Chave

Fontes, Modelos Generativos, Geração Automática de Glifos, Redes Adversárias Generativas, Auto-codificadores, Tipografia, Transferência de Estilo, Modelo de Difusão

Contents

Acronyms

CNN Convolutional Neural Network. 1

DCGAN Deep Convolutional Generative Adversarial Network. 8, 14, 18

EOS End of Sequence. 10, 16

ETA Estimated Time of Arrival. 44

FID Fréchet Inception Distance. 11

GAN Generative Adversarial Network. xvii, 1, 7, 8, 9, 12, 13, 14, 15, 17, 18, 21, 22

LSGAN Least Squares GAN. 22

LSTM Long Short-Term Memory Network. 9, 10, 11, 16, 17, 19

MAE Mean Absolute Error. 11, 22

MSE Mean Squared Error. 11

NLP Natural Language Processing. 9, 10, 15

OTF Open Type Font. 6, 7

PEGAN Pyramid Embedded Generative Adversarial Network. xvii, 13, 14, 15

RNN Recurrent Neural Network. 9, 10, 15

SOS Start of Sequence. 16

SVG Scalable Vector Graphics. 2, 3, 6, 16

TTF True Type Font. 6, 7

VAE Variational Autoencoder. 9, 16, 17

List of Figures

List of Tables

Chapter 1

Introduction

Historically, the manual design of new typographical fonts has proven to require a great deal of effort, even for skilled professionals [1]. Depending on the purpose and concept of the project, type designers need to define adequate styling features, such as width, height, weight, kerning, leading, and whether or not to use serifs or other adornments, among others. Moreover and even more complex is the process of designing congruent glyphs for all letters, numbers, and other symbols that compose the target writing system [2]. Designing and refining fonts often can take weeks or months after years of expertise. For instance, even for creating geometric fonts, minute optical adjustments are often needed. Therefore, due to the lack of time and resources, creating tailor-made fonts is many times not feasible in lower-budget design projects. And even with the availability of a large number of font families to end-users, editing minute details or ornaments might still be necessary in many design projects, e.g. in creating new brands, as these must be unique and distinctive.

For the past few decades, researchers have tried to tackle the aforementioned problems using computer techniques to automatically generate congruent fonts. Early research focused on statistical and pipelined approaches consisting of the decomposition and assembly of existing glyphs into skeletons, outlines, and parts with different levels of importance [3, 4]. While such approaches may suit more conventional fonts, i.e. designed according to more strict rules/topology, these cannot contemplate more unconventional typography, using unusual decorative elements or more experimental topology. Besides, such approaches have the shortcoming of being constrained and dependent on initial annotated data.

Meanwhile, the sharp rise in popularity of deep neural architectures in recent years has brought paradigm shifts to many fields of Computer Science. CNNs and GANs, in particular, have proven to be powerful architectures in pattern recognition problems and imaging domains. For example, these architectures can receive as input large quantities of glyph images and infer the styles and parameters that distinguish different fonts. As a consequence, they are capable of outputting visual results that are more believable than the ones produced by shallow solutions. However, the images produced by CNN models often come with perceptible degrees of noise and contour imperfections. Even in state-of-the-art work, this is a visible concern [5–7].

Furthermore, as fonts are typically represented via vector formats (so they can be scaled arbitrarily) and many of these models generate raster images, naturally, their outputs are not ideal for type designers to work with, as the glyphs will need to be converted into vector graphics before designers can properly refine them and program final fonts. Besides, there has been some work towards the creation of typefaces using vector formats such as Scalable Vector Graphics (SVG), which, again, seem to only work reasonably well for conventional typefaces, lacking in the generation of congruent typefaces when more divergent and experimental visual glyphs are given as an input. A good example of that is *Àdea* [8], a system that employs a genetic algorithm to generate innovative individual glyph designs (see Figure 1.1). As referred by the authors, although *Àdea* is able to generate distinctive glyphs, human designers are still left responsible for manually post-editing the outputs and creating all the remaining glyphs for a potential new font. Hence, we believe deep neural architectures, evolutionary systems such as *Àdea* and human designers can complement each other to ease the creation of innovative font designs.

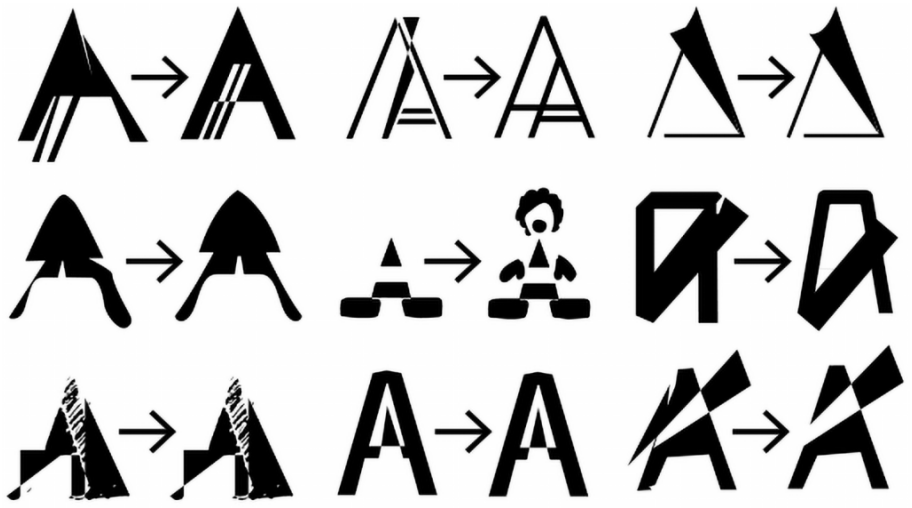

Figure 1.1: Glyphs generated by *Àdea* (left) next to user-edited versions (right). Figure originally from Lopes et al. [9].

In that sense, this dissertation focuses on the exploration and study of generative models for generating complete fonts (more specifically, sets of multiple Roman letters) out of one or a few input glyphs. Furthermore, we provide contributions to the current state-of-the-art and propose an approach that can be integrated into existing designing workflows by being able to generate congruent fonts in a scalable vector format out of less conventional input glyphs such as the ones generated by *Àdea*. Given what has been stated, the main objectives of this dissertation are summarised as follows:

1. Review the state-of-the-art of generative models that deliver complete fonts out of one or a few input glyphs;

2. Collect relevant datasets that can be used in the training and validation of models with varied architectures;

3. Develop a system, supported by generative models, capable of producing fonts, in an SVG-like format, with style and design choices assimilated from glyph input references;

4. Analyse the performance of the developed generative system and compare it to other state-of-the-art solutions for font generation.

1.1 Contributions

The work done in the scope of this project led to the following contributions:

- Comprehensive review of the state-of-the-art in font generation through generative approaches;
- Replication and implementation of generative methods based on state-of-the-art methods and techniques;
- Identification of the strengths and shortcomings of the implemented models as well as the test of additional enhancements/modifications;
- Proposal of adequate performance metrics and execution of in-depth quantitative testing;
- Proposal of a combined approach to tackle the suggested goals for this project;
- Development of a web application that demonstrates and highlights the applicability of the approach devised in this dissertation.

1.2 Document Outline

In this first chapter, we introduced the glyph generation problem and highlighted its challenges. Additionally, we contextualized the project of this dissertation with preceding works and outlined its main objectives and requirements.

Chapter 2 briefly introduces type design and summarizes key concepts in machine learning required to understand the state-of-the-art in the automatic generation of typographical fonts.

In Chapter 3 we propose a novel approach to tackle the referred problem alongside the proper rationale behind it. Furthermore, the necessary tools to materialize our approach, which include models, data, and validation metrics, are introduced in this section as well.

Chapter 4 encapsulates all the experiments conducted with the models developed throughout the second semester and introduces our contributions to the font generation task.

In Chapter 5 we present a web application designed to showcase our approach and the potential for its real-world usage.

Chapter 6 describes the methodology and schedule followed in this dissertation. Lastly, we conclude with a sum up of the work done and pinpoint future efforts in Chapter 7.

Chapter 2

State of the Art

In this chapter, some key concepts in typography and type design will be introduced to establish a foundation for understanding subsequent discussions about the quality and type of outputs produced by generative models responsible for constructing new glyphs and even complete fonts.

After that, a set of model architectures and existing techniques are briefly explained and the relevancy of each one for the research field of this dissertation is further discussed in section 2.2. These are complex models, frameworks, and techniques often used in state-of-the-art approaches for glyph generation hence the inclusion of an introductory section.

Finally, in Section 2.3, a comprehensive review of the current state-of-the-art approaches for glyph generation is presented alongside some discussion about the comparative strengths and shortcomings of each method.

2.1 Type Terminology

In the typography field, the term "font" refers to a single set of characters instantiated from a typeface in a given style (e.g. bold, regular, light, etc) while "typeface" denotes a font family: a collection of fonts closely related to each other, typically, designed simultaneously by the same authors [2].

Commercial fonts are typically crafted by experts, referred to as type designers, who are able to manipulate a multitude of glyph variables in order to meet their customer's needs. Although type design is simultaneously an art form and a complex research field on its own, according to Cheng [2], the following can be considered the main variables that help distinguish different font families (please note the terms may slightly vary from author to author [1, 10]):

- **Serifs and terminals:** terminals occur at the end of character strokes and may be followed by smaller, perpendicular strokes called serifs. A font making use of serifs is called a serif font;
- **x-height:** the height of lower cased letters;

- **Weight:** the "thickness" or boldness of a character;
- **Spacing/kerning:** the space between individual characters. While spacing determines the overall length between words in a text block, kerning specifically adjusts the distance between certain letter pairs, like **av**, to create a more natural appearance. Kerning is integral to the font identity, whereas spacing can be adjusted in software by the writer as desired;
- **Contrast:** ratio between vertical and horizontal stroke thickness.

While there are many other styling attributes that contribute to font uniqueness (e.g. the ones visible in Figure 2.1), the aforementioned aspects may suffice in helping readers to grasp the employed terminology and the visual differences in glyph quality presented in the imagery of Chapters 2 and 4.

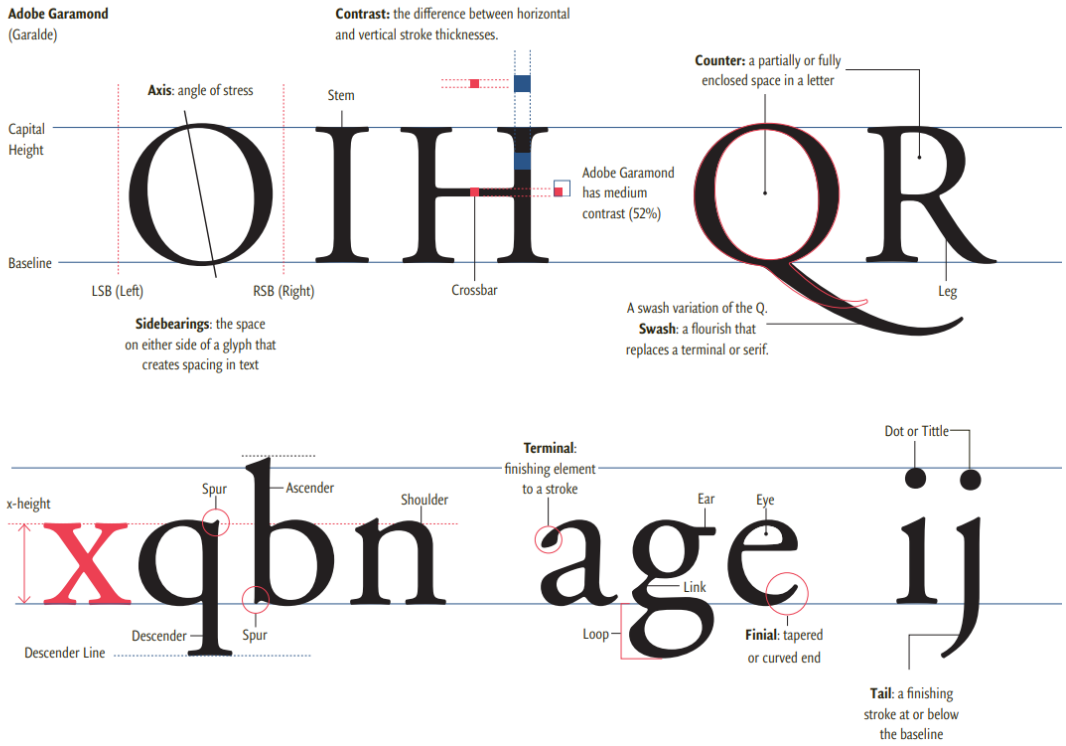

Figure 2.1: Type terminology, by Cheng [2].

Besides identity and styling, fonts must be designed to be used and presented in a multitude of media. Therefore, modern fonts are typically recorded in vector formats. Unlike raster images that are defined by matrices of predefined dimensions, vectorized objects can be scaled to fit any screen or print size without suffering any loss of information while maintaining compact file sizes. The True Type Font (TTF) and Open Type Font (OTF) are two examples of popular formats for representing fonts in Operating Systems and software applications.

Scalable Vector Graphics (SVG) is an XML-based vector image format developed by the World Wide Web Consortium that became an open standard for 2D graphics representation [11]. Using SVG, designs can be represented by an ordered

sequence of paths (points, lines and bézier curves) and/or composed of primitive shapes such as points, lines and polygons. This is also a popular format for designing fonts as the designs can be easily turned into final font files such as TTF or OTF.

2.2 Generative Models

Generative models are often defined as models that capture the joint probability $p(x, y)$ of a set of original samples X with labels Y in order to create new, unobserved ones following that distribution [12]. Unlike discriminative models that attempt to draw boundaries in the data space to learn the posterior distribution $p(y|x)$, generative models tackle more difficult tasks in capturing more information about the training data. There is a multitude of architectures that follow this broad definition, including the following that will be explored in more detail in this section to better contextualize current state-of-the-art systems mentioned later on:

- Generative Adversarial Network (GAN) models
- Variational Autoencoders
- Auto-regressive models
- Diffusion Models

Generative Neural Networks (GANs) [13, 14] are a subset of neural networks designed to generate new data following the underlying distribution of training samples.

A GAN is traditionally composed of two networks: generator and discriminator (see Figure 2.2). The first module attempts to generate new artificial samples based on an existing dataset while the discriminator's responsibility is to discern between real and generated samples fed to it. The loss of the generator depends on its capability to fool its counterpart, therefore, enabling unsupervised learning.

$D(x)$ denotes the output probability, computed by the discriminator, that x came from the training set rather than created by the Generator. $D(x)$ should be high when x comes from the original data and low otherwise. Meanwhile, $G(z)$ represents the output sample of the generator given a latent vector z as input. $D(G(z))$ is the probability of generator output coming from the real-world distribution. During training, both the generator and discriminator follow a min-max game with function $V(D, G)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \backsim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \backsim p_z(z)}[\log 1 - D(G(z))] \tag{2.1}$$

where G and D minimize and maximize this loss function, respectively.

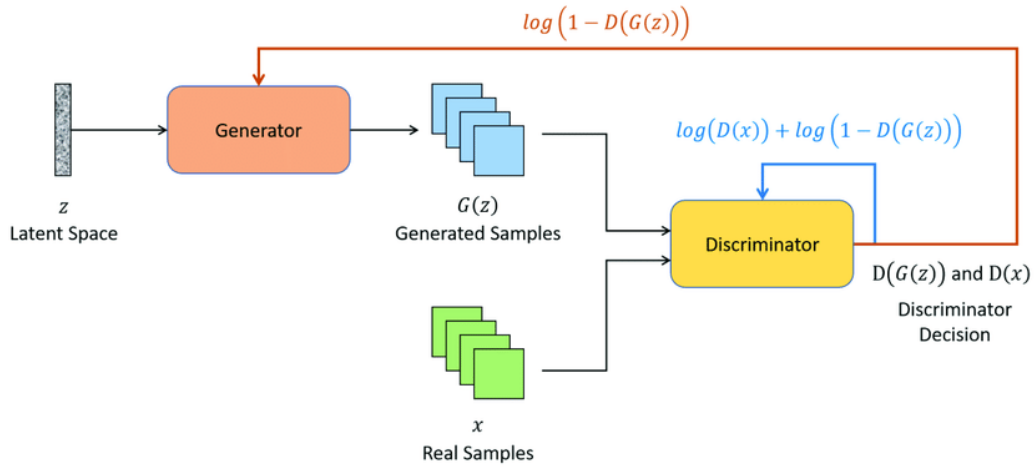

Figure 2.2: GAN Architecture, from Vint et al. [15].

GANs found notoriety in image generation problems by incorporating deep learning architectures and convolutional layers in both generator and discriminator [16]. To this end, Deep Convolutional GANs (DCGANs) were initially introduced in 2016 by Radford et al. [16].

DCGANs are particularly useful if one considers the font generation problem as being part of the broader image generation field of research. Although such networks are harder to train due to increased depth, multiple research projects were able to create credible typefaces by giving the label of the desired glyph and vectors of high-level features, such as boldness, contrast, and use of serifs, as input to these network models [5–7, 17]. This approach surpasses other methods by allowing the decoupling between two distinct phases:

1. Feature Extraction - phase in which features with lower dimensionality and/or humanly interpretable are extracted from a few user-made glyphs;

2. Typeface Generation - GAN-supported system attempts to generate the remaining characters taking into account the output of phase one.

Other than generating new samples following underlying probability distributions, certain generative models can also be used to translate glyph inputs to latent spaces with fewer dimensions, effectively fulfilling the goal of the first stage (i.e. Feature Extraction). Such is the case of autoencoders and variational autoencoders . Autoencoder networks [18] are used to map input data to a latent space in an unsupervised manner and are composed of two components: the encoder, which consists of a series of layers with decreasing number of nodes, and the decoder, which mirrors the encoder and attempts to generate the initial input based on the features outputted by the first component.

During training, the parameters of the autoencoder are updated by the reconstruction error measured between the original data and the data generated from the latent representation created by the encoder. In general, autoencoders find

new representations of the initial data and are mainly used in dimensionality reduction, feature extraction, image denoising and image generation tasks.

Variational Autoencoders differ from the initial autoencoder model by ensuring the encoder network maps the initial input to a continuous latent space fit for generative purposes [19, 20]. The loss function of VAEs shelters the initial reconstruction error and an additional regularization term responsible for ensuring that any point sampled from the latent space leads to a meaningful output by the decoder.

Although VAEs may not be ideal for image generation tasks as standalone models [21], they can be powerful tools capable of encoding images into latent spaces with fewer dimensions that can be manipulated by users and fed to other generative approaches such as GANs and Transformers. These smaller representations can correspond to a subset of crucial type parameters that define the design of a font family.

While GANs and Autoencoders have been successful in generating high-quality raster images, more recent approaches to glyph generation problems using recurrent neural networks have started to emerge. These models are now starting to be used to meet the requirements of technical users in demanding outputs that can be further manipulated and fine-tuned to their liking.

Long short-term memory networks (LSTMs) are a type of Recurrent Neural Networks (RNNs) introduced by Hochreiter and Schmidhuber [22] and designed to deal with the issues of long-term dependencies in standard RNNs. Most modern variants [23, 24] are composed of cell blocks arranged in series that compute an output (h_t) based on the input at time t (X_t) and the state of the previous cell (C_{t-1}). At each cell block, the previous cell state is altered — information is both added and deleted — based on multiple learnable layers, enabling the whole network to learn long-term dependencies between input samples (see Figure 2.3).

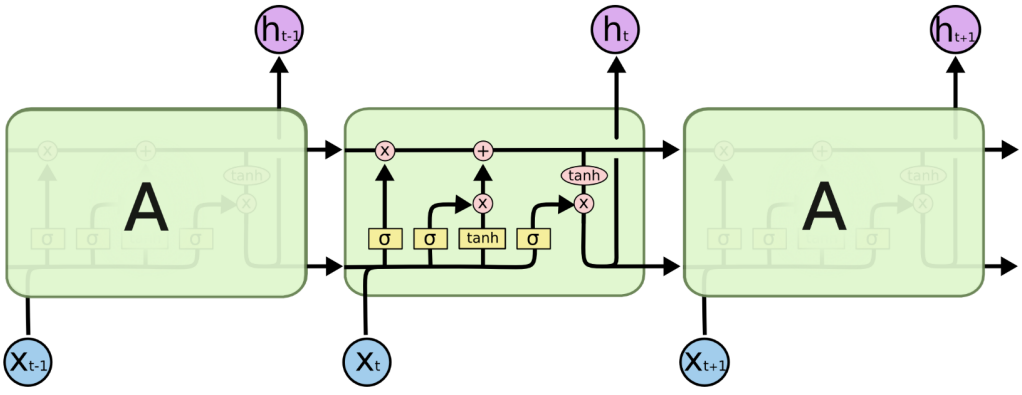

Figure 2.3: Typical LSTM module expanded. The cell state of the previous block is modified by the output of several layers. In turn, these layers take into account the current and previous inputs fed to the network.

LSTMs are widely used in Natural Language Processing (NLP) tasks, speech recognition, and ECG signal analysis where data is typically represented through

discrete time series — every input sample is highly dependent on previous data. The font generation problem can take advantage of LSTMs and other RNNs by shifting from an image generation task to the generation of SVG sequences composed by a defined set of primitive commands such as *moveTo*, *lineTo*, *cubicBezier* and *EOS*. Doing that, the advantages of using convolutional layers to easily extract imaging patterns and font styles would be traded by the ability of LSTMs to output vectorized representations that are scalable and much more useful for designers to refine.

In recent years, LSTMs have begun to be replaced by Transformers, which have shown to be more robust in the majority of NLP tasks. The Transformer was originally introduced by Vaswani et al. [25] and since then has quickly been established as the leading architecture for sequence-to-sequence tasks in the NLP domain such as translation, summarization, question-answering, speech recognition, etc.

Transformers outperform traditional recurrent networks by arranging themselves in encoder-decoder-like architectures, similar to autoencoders, and by incorporating the concept of self-attention, where each element of a given text sequence holds an importance scalar to every other token in that sequence. The self-attention mechanism present in several of its layers effectively allows for global dependencies to be captured, unlike RNNs which are limited to localized relationships. In contrast to RNNs, both during training and inference, transformers take as input all the sequence tokens at once and the training procedure allows for the parallelization of the decoder optimization. Currently, two of the most prominent transformer implementations are BERT [26] and GPT [27].

Lastly, novel approaches to image generation have begun to emerge through the use of diffusion models that outperform prior techniques [28–31]. Such approaches make use of the same neural architectures employed in GAN generators, CNNs, and autoencoders but are fundamentally different in terms of their objective function. These models are trained to predict and eliminate artificially added noise in training samples. Consequently, they can generate new images from pure Gaussian noise using conditional variables such as target class and style. During the training phase, each sample undergoes a diffusion process in which conditional Gaussian noise is introduced to each pixel. The noise scheduler determines the extent of noise added using the following formula:

$$q(x_t|x_{t-1}) = N(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{2.2}$$

where x_{t-1} represents the original sample while x_t is that same sample with a certain level of additional noise. The value of β_t controls the magnitude of noise, usually linearly sampled from the interval $[0.0001, 0.02]$ over $T = 1000$ steps. A higher timestep t corresponds to a larger β_t, resulting in more noise. This formula can be iteratively applied to obtain the noisiest version x_{1000} of the original image x_0. As the training progresses, the network, typically instantiated by a UNet [32], learns the reverse process by predicting the added noise, which can then be subtracted to approximate the original sample as $x_{t-1} \approx x_t - \epsilon_0$. During testing, this backward/diffusion process is repeated for $T = 1000$ timesteps to generate

$\bar{x}_0$, commencing with Gaussian noise fed into the network alongside some conditional embeddings.

To evaluate and compare the performance of various generative models, several metrics are readily available, one of which is the Fréchet Inception Distance (FID) [33]. The goal of generative models is not merely to replicate the original inputs precisely, and hence, FID was specifically devised to measure the similarity between the distributions of real and generated image sets. Lower FID values translate to better image quality on generated outputs. It measures the distance between the activations calculated using one of the final layers of the Inception Model [34] of real and fake data. These activations are assumed to follow multivariate Gaussian distributions (m_R, C_R) and (m_G, C_G), respectively, and the FID value is calculated as the Fréchet distance between the means and covariances of these two distributions:

$$\begin{aligned} FID(G,R) &= d^2((m_G, C_G), (m_R, C_R))) \\ &= ||m_G - m_R||^2 + Tr(C_G + C_R) - 2\sqrt{C_G * C_R} \end{aligned} \tag{2.3}$$

Since FID operates on image representations, the glyph outputs of LSTMs and Transformers, typically expressed through sequences of SVG instructions, need to be translated to raster forms in an intermediate step before submission for evaluation. Other possible metrics for comparing real and artificial font images include Mean Squared Error (MSE) and Mean Absolute Error (MAE), which directly compute the differences between individual pixel values and output the corresponding mean across each image pair. SSIM is a metric commonly used as well for performance comparison in image generation tasks. In SSIM, luminance, contrast, and structure values are computed for two images and then combined according to weights α, β, γ:

$$SSIM(G,R) = l(R,G)^\alpha + c(R,G)^\beta + s(R,G)^\gamma \tag{2.4}$$

In contrast to FID, the Structural Similarity Index (SSIM) scores are presented on a scale of -1 to 1, with a value of 1 indicating complete similitude between the two images being compared.

2.3 Font Generation Through Modern Generative Approaches

Automatic font generation is a long-studied problem with a wide range of solutions being proposed throughout the years. One of the earliest systems created by Suveeranont and Igarashi [3] allowed users to compute a new design by providing a single glyph reference in what can be considered the first one-shot approach. In this system, a small database of template fonts was compiled where each glyph is represented by the skeleton and outline, thus capturing their structural topology. When the user provides a given glyph with stylistic choices to be reproduced in the remaining characters, these topology features need to be

drawn by the user so that the system can fetch a unique combination of examples found in the database that when blended together match the user input. After finding a proper match, the topology is equally blended for the reaming glyphs, thus creating a new font. The main fallbacks of this approach are the reliance on the annotated dataset with topological information, hence why the original database contains only 32 examples, and the incapability to generate novel designs since the outputs are a combination of a very small set of already existing fonts. The model is not flexible enough to accommodate stylized/ornamented fonts which contain wildly different topological elements not found in any other design. Even the authors question its efficacy in covering two distinct lowercase variations of the same letter (e.g. *a* vs *a*).

Another classical approach can be found in the rule-based application developed by Cunha [4]. After users provided some initial glyph designs, this application was capable of assembling the remaining glyphs by employing a set of rules based on a pre-established relationship scheme between lowercase Roman letters. Not only the users had to provide multiple vectorized glyphs, but they also had to manually identify specific typography components for each provided glyph through the selection of SVG control points. These components, such as stems, descenders, and loops, were then reused in the automatic generation of the characters that were not initially provided by users according to the pre-established rules (e.g. the lowercase m can be obtained through the assembly of shoulder and stem from lowercase n). In spite of the generation process behaving in a very predictable way, this system was limited by the initial set of fixed rules, making it unreliable in situations where such instructions were not applicable. Furthermore, users were required to have prior typography knowledge in order to properly annotate inputs. While this requirement is acceptable in systems conceived to assist type designers in the early design stages, it becomes unattainable when targeting the general public.

In the opposite sense, the *Àdea* system developed by Lopes et al. [8] was able to generate an unlimited supply of innovative glyph designs through the implementation of an evolutionary algorithm. In this system, a population of designs for a single Roman Character is evolved until the user obtains a satisfactory outcome. Each individual is encoded by an SVG path consisting of *Line* and *Move To* instructions limited to a 100x100 pixel canvas and initialized either by a random Google Fonts typeface or a random SVG sequence. Given the stochastic nature of the generation process, the system was able to achieve stylized glyphs composed of unconventional shapes. However, users are unable to control the design process and are limited to choosing the final output from the evolved population. Another significant drawback of this approach is its limitation to evolving a single character at a time. Therefore, if users want to incorporate the system's design into their work, they must manually design the remaining characters themselves.

Xie et al. [35] were able to successfully generate rasterized Chinese characters and thus achieving low FID and MSE scores in unseen samples. Their GAN-based system requires a content image, containing the target character, and a style image carrying an unrelated glyph with distinct design features and desired style (see Figure 2.4). These inputs are then encoded to latent vectors and fed to the

generative network that transfers the desired style into the first character.

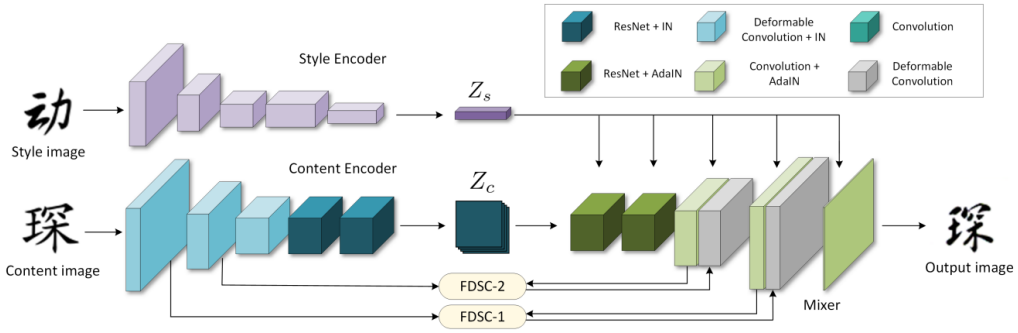

Figure 2.4: Overview of the generator from Xie et al. [35]. The latent vectors are easily accessible and the class of the output is only restricted by the content images seen during training.

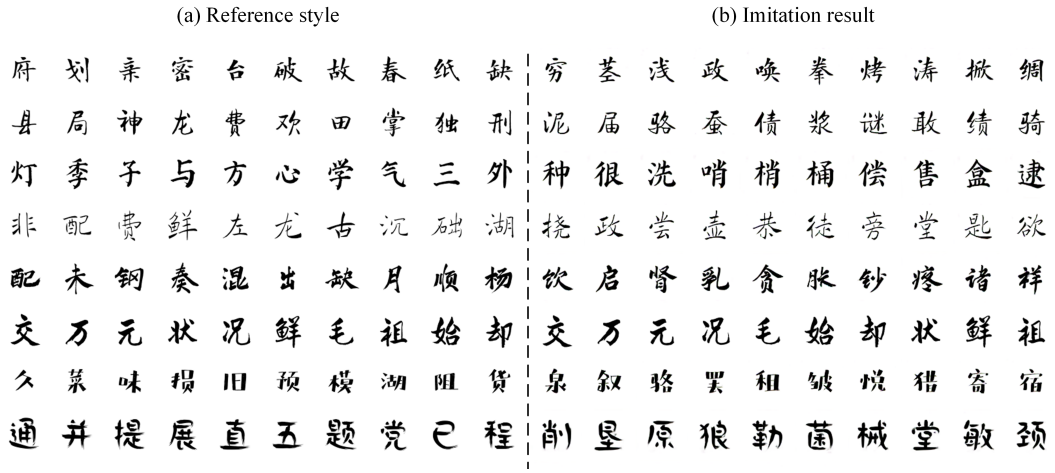

Figure 2.5: Result samples from Xie et al. [35]. The network attempts to apply the styles seen on the left side to other characters.

The inclusion of the content encoder in the final generator architecture appears to be redundant given that other related approaches rely on decoders conditioned on the output glyph labels, which effectively enforces the generation of images in those desired classes [5, 36]. Nonetheless, the employment of style transfer techniques is an advantage over other solutions because, instead of solely relying on abstract/latent information such as style or label encodings, the model is guided by image references in the generation of the output. According to Figure 2.5, this method is able to generate very clean Chinese glyphs with an identical style to the ones given as reference. To corroborate the robustness of this method is the fact that the neural network employed here was incorporated in posterior state-of-the-art models such as He et al. [31].

In their work, Sun et al. [7] integrated an encoder-decoder network into the generator of their GAN framework for creating Chinese fonts. The proposed framework, known as PEGAN, aims to generate multiple styles for a given Chinese

glyph by utilizing an encoder-decoder module inspired by the U-Net [32] architecture. Each encoding layer takes both the activation of the preceding layer and the downsampled source image as input (via the refinement connection). The decoding layers receive the output of the preceding layer along with the encoding of the mirrored encoding layer (via the skip connection). The results obtained from this approach exhibit slight improvements compared to the zi2zi method, effectively addressing the reported stroke completion issues present in zi2zi. The zi2zi method developed by Tian [6] is a competing approach for generating Chinese calligraphy that employs the GAN architecture and is able to manipulate latent style vectors as well. The primary distinction between these two approaches lies in their generator architecture and loss implementation.

Figure 2.6: Generated samples using the PEGAN method from Sun et al. [7] compared to ground truth references.

This approach shows that minor improvements made to already well-established architectures can, in fact, enhance the results in targeted research domains and proposes the use of accuracy values computed with an independent classification model as a possible quantitative metric in font generation tasks. However, the paper's visual results are relatively limited in quantity and may not possess the same visual impact as other works, such as Azadi et al. [5], particularly regarding the complexity of ornaments (see Figure 2.6). Nevertheless, the generated glyphs showcase in the original paper systematically exhibit well-defined edges and clear variations in boldness, contrast, and stroke endings.

Out of all the solutions that rely on raster images as final outputs, Azadi et al. [5] achieved the most promising results with their Glyph Network by incorporating up to six ResNet layer blocks in its DCGAN generator and using stacks of multiple pre-made glyphs as input to generate the remaining characters.

As seen in Figure 2.7, this model excels at reconstructing complex details such as terminals, serifs, and outline patterns present in the initially observed subset of characters and translating them to the unseen ones in a coherent manner (e.g. the ornaments in the H crossbar are preserved in B and C glyphs). Although it is out of our project's scope, the authors also employed a second network capable of filling the results of the first network with color gradients and removing noise

Figure 2.7: Samples generated with Glyph Network [5]. This model was trained on a dataset of 10k fonts of unique designs, which resulted in generally reliable results, although some imperfections and noise are noticeable.

artifacts to improve the quality of the outputs even further.

A significant limitation of this method is the requirement to obtain multiple predesigned characters to achieve satisfactory results. In the original paper, the authors used 6 images per font as an input to achieve the quality depicted in Figure 2.7. Our replication results exposed in Chapter 4 confirm a significant performance decrease when fewer initial characters are provided to the model. Another disadvantage of this work over other approaches [7, 35] is its incapacity to create lowercase glyphs, i.e. can only generate the 26 capitalized western letters and requires major architectural changes if one desires to expand this subset.

Although deep convolutional networks, such as the Glyph Network [5], PEGAN [7] and others [6, 35], are capable of producing artificial typefaces with complex ornaments and features, the majority of the implementations are plagued by outputs of low resolution with perceptible stroke imperfections. These images alone are not ideal to be used as final fonts given that they do not scale well to higher dimensions and lack the rigorous quality demanded by type designers. Much more post-processing and manual work would be required before arriving at designs with satisfactory quality.

To overcome the limitations of raster-based models, there is already research in image-to-vector translation techniques [37–42]. One notorious work is the LIVE framework developed by Ma et al. [42] that is capable of translating images of any domain such as Emojis, handwritten characters, and clipart images to vector representations controlled through a number of bézier paths pre-defined by end-users (see Figure 2.8). Since this framework is model-free, it is not constrained to specific training datasets and can be used in a variety of applications, including the translation of raster fonts generated by the GAN models previously mentioned in this section.

Besides GAN-based approaches, the alternative paradigm in font generation resides in applying knowledge acquired in the Natural Language Processing (NLP) domain by using RNNs and Transformers to process and generate sequences of

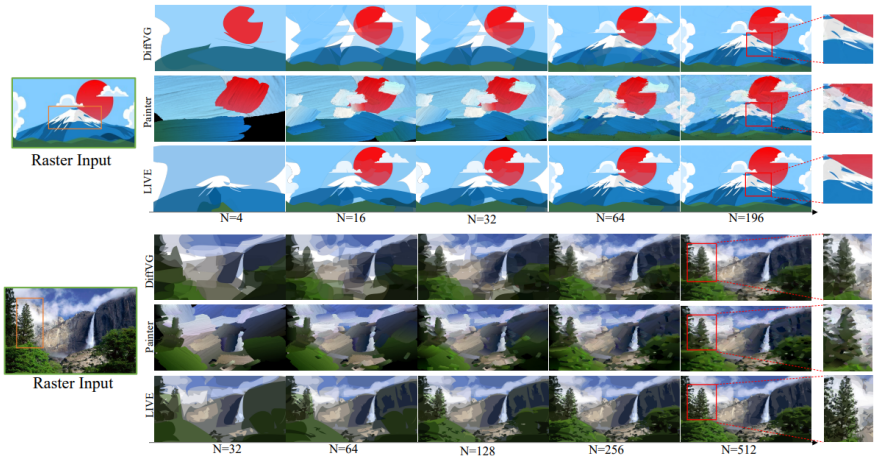

Figure 2.8: Progressive translation of complex input images to vector representations using the LIVE framework [42]. The number of allowed vector paths is progressively increased from left to right until visual similarity is achieved.

SVG-like commands representing individual characters.

Aoki and Aizawa [43] developed a Transformer for generating Chinese characters that attains consistently better results when compared to other parallel approaches [44]. In this architecture, the encoder extracts feature vectors from a style reference while the decoder is given those vectors along with a content reference to output the final glyph vector consisting of a sequence of the SVG commands MoveTo, LineTo, Cubic Bézier Curve, Close Path, <SOS> and <EOS>. While this implementation can be used for western alphabets, it is limited to the generation of stroke outlines and is incapable of filling the glyphs with textures like the ones seen in Figure 2.7.

When compared to raster approaches, where noise found in individual pixels does not have a perceptible impact on the overall model performance, this system requires much more technical control and pre-processing over the samples given to it. Fonts in SVG format can typically be composed of long, complex sequences of instructions that need to be simplified alongside the normalization of each command argument while still ensuring the initial glyph structure is not lost.

To the best of our knowledge, the contributions of Lopes et al. [36] are to date the largest in typography generation tasks. By coupling a convolutional variational autoencoder with a network of stacked LSTMs, the authors were able to create a system that not only produces customizable glyph vectors but that also is interpretable and explainable as in no other approach reviewed.

By training the VAE with a large-scale dataset of 14M font characters, it was possible to learn the latent representation of any glyph, which can be manipulated and exploited to alter the stylized glyph output in a nuanced and controlled manner, as seen in Figure 2.10. Furthermore, the usage of the autoencoder network allows the partitioning of the font generation problem into two smaller tasks. Unlike in

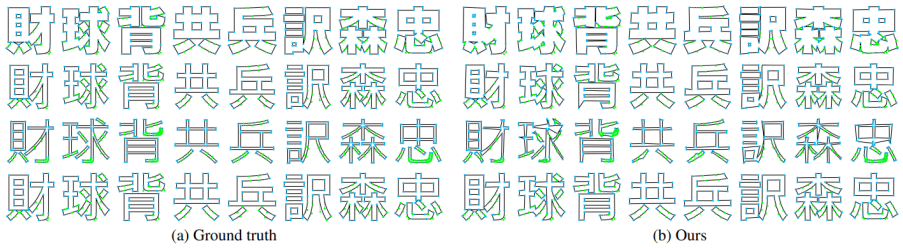

Figure 2.9: Reconstruction examples by Aoki and Aizawa [43]. Although the generated samples do not contain any path collisions or defects, they lack rounded edges and show limited variation in type parameters if boldness is excluded.

previous methods, the LSTM network is not simultaneously responsible for the identification of key features in source glyphs and the generation of new ones based on those features, leaving the first duty for the VAE.

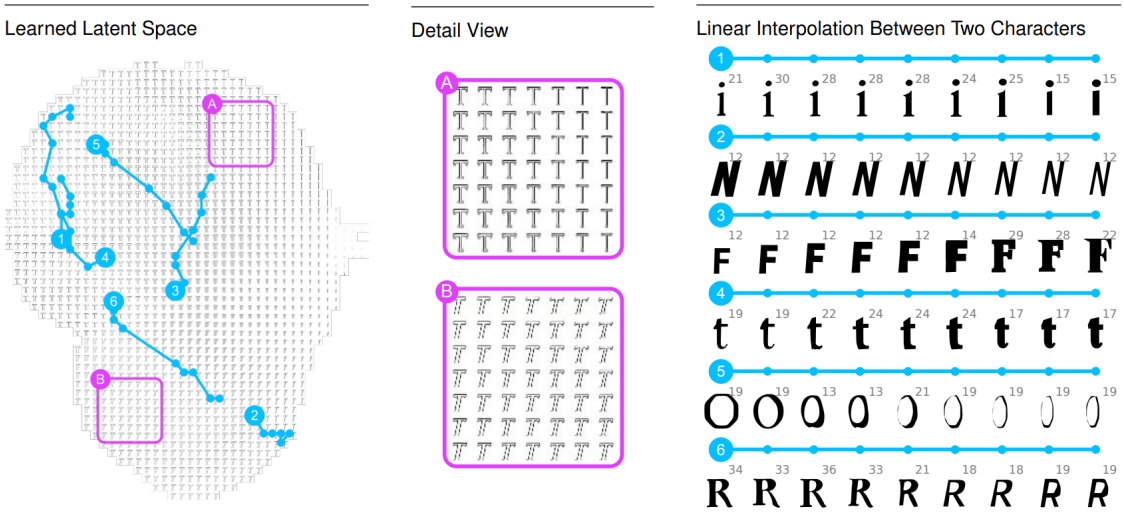

Figure 2.10: Manipulation of latent font representations done in the SVG-VAE project [36]. Although each latent space vector is generally multidimensional, it can be projected to the 2D space for human readability and be further manipulated by exploring neighboring values.

One limitation of the dataset used in this project is that it only includes web-oriented fonts that do not have highly decorative styles like the dataset in Azadi et al. [5]. These fonts only vary in basic features like boldness, tilt, and serifs. Thus, it remains uncertain how well the model presented by Lopes et al. can handle more intricate glyphs.

Aside from systems designed to output vector representations, much of the noise artifact issues inherent to GAN models seem to be surpassed in recently developed diffusion networks such as He et al. [31]. Although diffusion networks specialized in glyph generation are yet scarce, such paradigm has proven its ability to achieve comparable performance scores and effectively addresses problems related to stroke imperfection and edge jitteriness that were prevalent in earlier

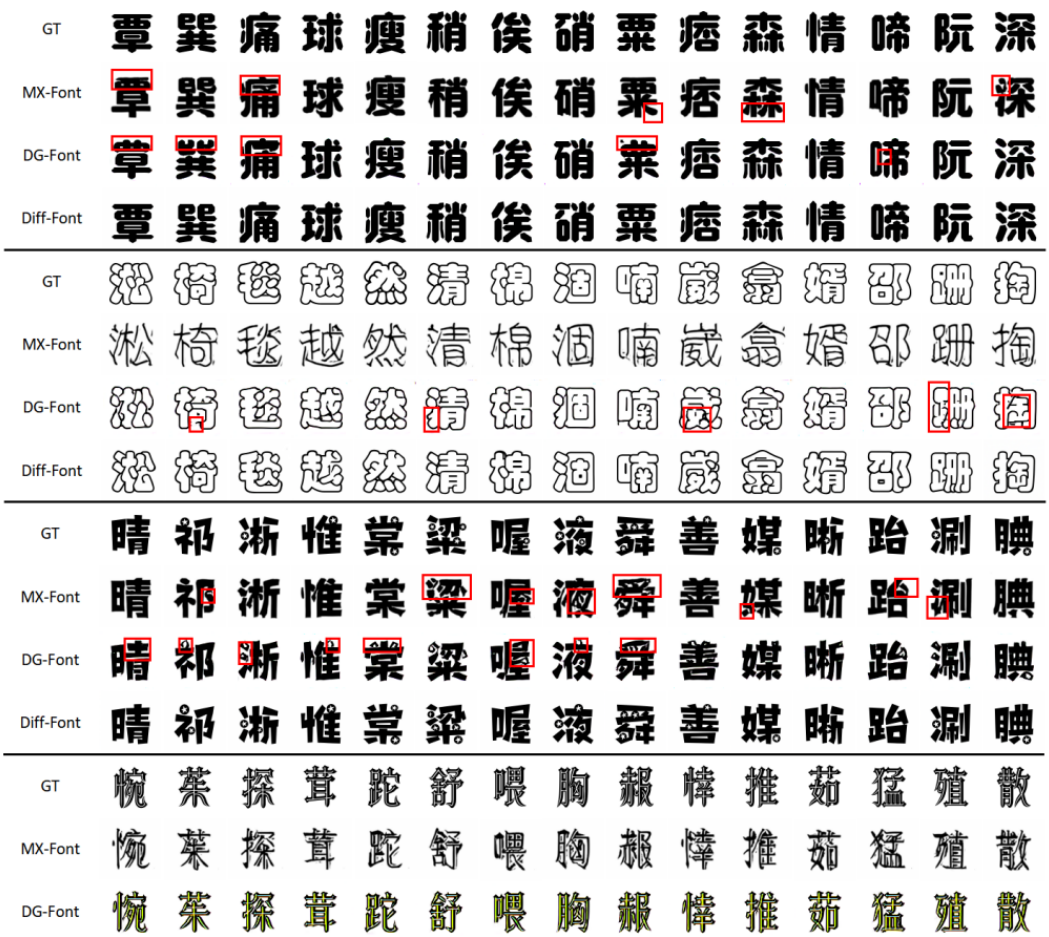

Figure 2.11: Visual outputs provided by He et al. [31] that show the ability of their Diff-Font model to improve on previous state-of-the-art models.

GAN methods. Some of these improvements are evident in the visual results shown in Figure 2.11. Sections 3.2 and 4.3 provide a comprehensive analysis of the Diff-Font model, developed by [45].

Summary

To summarize, a diverse range of generative methods has been utilized for font and glyph generation. Each method presents its strengths and drawbacks given that each one is specialized in a specific writing system, output format, or the type and number of inputs required to function properly. While classical approaches that exploit glyph typology rules are computationally less complex than deep generative models, they struggle to complement the design of a wider variety of fonts and often require user-annotated data [3, 4].

A large majority of modern approaches rely on the Deep Convolutional Generative Adversarial Network (DCGAN) architecture [16] to allow non-technical users to generate complex designs based on a single or a small set of input glyphs, often with very minimal effort [5–7, 35, 46–49]. The main drawbacks of this paradigm are the image noise and imperfections that plague the generated out-

puts and the fact that these approaches are only capable of outputting raster representations. These outputs need to be corrected and translated to vectorized representations later on, leading to unexpected burdens imposed on users. Furthermore, the majority of these works target Eastern writing systems such as Japanese and Chinese and do not disclose achieved performance results through the measurement of quantitative metrics, leaving the potential for its applicability to Western glyphs unknown.

The alternative paradigm relies on LSTMs and Transformers to generate vectorized fonts [36, 43, 44], eliminating prior noise issues altogether. These approaches are often more complex and required larger quantities of pre-processed to achieve satisfactory results, which is systematically not provided by the original authors in order to properly replicate results. Recent advancements in the development of Vision Transformers [50, 51] and Diffusion Models [28–30] also begin to surpass GAN-based approaches in broader imaging domains but their application in font generation is yet reduced [31].

Another observed trend in font generation systems is the adoption of encoder-decoder architectures that enables better user interpretability of outputs in machine learning models [35, 36, 48, 50, 52]. The usage of Autoencoders and Variational Autoencoder structures allows the extraction of style from input glyphs and posterior representation in 1D vectors that map human-readable type parameters such as boldness and contrast. This is an advantage over other approaches, given that it allows users to slightly modify their designs mid-generation process in an easy way.

Chapter 3

Approach

As previously mentioned, the goal of this project is to devise a system capable of generating stylized fonts based on as few input glyphs as possible. Besides, the output of the system is desired to be easily editable by type designers such that the time spent fine-tuning the output must be significantly lower than the time required to design each glyph from start to finish. Furthermore, we aim to make the system available online through a user interface for it to be as accessible as possible to type designers and practitioners with no coding expertise.

Given the lack of open-source datasets containing vectorized Roman glyphs and the abundance of research in the image generation domain, as reported in Section 2, we opted to partition the original problem into two smaller tasks: give the primary focus on generating complex glyphs with state-of-the-art generative approaches in the raster domain and subsequently transforming those artifacts in vectorized representations using general-purpose systems such as the LIVE system [42]. An outline of our approach to the glyph generation problem is outlined in Figure 3.1.

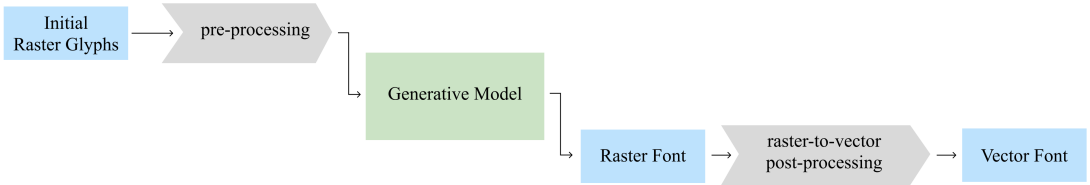

Figure 3.1: Overview of the proposed system comprised by the main generative model, capable of outputting raster images with realistic ornaments, and a secondary component responsible for the translation of such outputs to the vector domain.

Some of the GAN architectures presented in Section 2, which showed the most promising results and potential for improvements, were replicated, modified, and subsequently exploited to achieve the results shown in Section 4. The primary objective of this chapter is to thoroughly analyze the GlyphNet and DiffFont models, elucidating their potential in contributing to the goal of this dissertation. Furthermore, it aims to establish a foundational understanding neces-

sary to comprehend the crucial modifications we propose for the loss functions and inputs of these state-of-the-art techniques. These proposed changes result in trained models exhibiting enhanced overall performance.

We first summarize how the original Glyph Network operates and establish a baseline training setup in Section 3.1, followed by Section 3.2 where Diff-Font [31] is presented. This is a diffusion model used as a guiding architecture for our own diffusion approach in later stages of experimentation. In Section 3.3, we provide an overview of the dataset *Capitals64*, which is utilized in all experiments. Additionally, we present new data specifically designed for addressing the glyph generation problem and serving as a benchmark for future approaches. Finally, Section 3.4 introduces the metrics employed to conduct the quantitative analysis and comparisons performed on the presented models.

3.1 Glyph Network

The Glyph Network, proposed in Azadi et al. [5], presents a modified version of the GAN framework that exhibits the capability to produce more complex fonts when compared to the remaining raster methodologies discussed in Section 2. The adoption of this network over others was also influenced by its application to the generation of Roman characters in the original work. Nevertheless, the initial paper lacks a comprehensive quantitative performance assessment, making it an ideal choice for replication studies before advancing with modifications to some of its core architecture components.

The Glyph Network includes three key modifications to the original GAN [14]. Firstly, the input to the generator (G1) consists of a 26-channel font stack, with each channel corresponding to a unique grayscale glyph in a 64x64 tensor. Some of these input glyphs are hidden, meaning that their corresponding channels are filled with null/zero values. The generator's goal is to reconstruct all 26 channels, including those that are not included as the input subset.

Secondly, the Glyph Network includes not only a global discriminator (D2) that determines whether a given stack is fake or real but also a smaller network that acts as an auxiliary discriminator for each individual glyph. The network computes the probability of each glyph in the stack being fake. These two loss components output LSGAN losses that are combined with the L1 Loss during the backpropagation phase, which leads to the third architectural change to the GAN framework.

In addition to the typical discriminator loss, the generator optimization process is also influenced by direct penalization values computed between the generated output and the ground truth in the form of Mean Absolute Error (MAE) (also known as L1 norm). The full loss function is defined as follows:

$$\begin{aligned} \mathcal{L}(G_1) &= \lambda_1 \mathcal{L}_{L_1}(G_1) + \mathcal{L}_{LSGAN}(G_1, D) \\ &= \lambda_1 \mathcal{L}_{L_1}(G_1) \\ &+ \mathcal{L}^{local}_{LSGAN}(G_1, D_1) + \mathcal{L}^{global}_{LSGAN}(G_1, D_2), \end{aligned} \tag{3.1}$$

where G1 denotes the generator network, D1 and D2 refer to the local and global discriminators respectively and λ_1 is the weight of the L1 loss component. A broad illustration of this architecture can be seen in Figure 3.2.

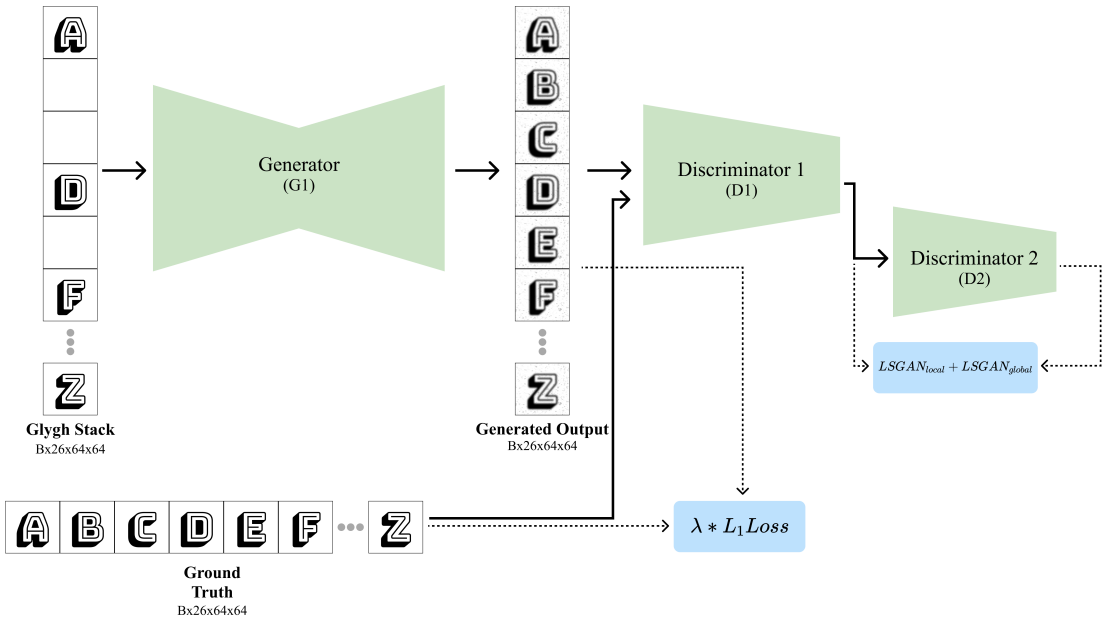

Figure 3.2: Architecture of Glyph Network. The generator receives some glyphs of the desired font and tries to generate the remaining ones by preserving the style. Discriminator 1 is tasked with discerning between real and fake samples while discriminator 2 does the same on whole stacks.

3.2 Diff-Font

Diff-Font is a conditional Diffusion Model developed by He et al. [31] for the generation of Chinese and Korean characters in 64×64 images. It is composed of two modules: a feature extractor that encodes the feature attributes of a given character A into a latent space, and the diffusion network which uses feature attributes of A and the class embeddings of character B to generate the desired character B from Gaussian noise.

Figure 3.3 demonstrates the incorporation of feature attributes obtained by a pre-trained style encoder developed for DG-Font in Xie et al. [35] in the proposed approach. These attributes are concatenated with the class embeddings representing the target character in a 1D vector of 512 bits. The resulting concatenation is then added to the activation values of several layers within the diffusion network. The diffusion network, based on the U-Net architecture, comprises downsampling ResNet blocks [53] followed by upsampling blocks and follows the diffusion training process proposed by Ronneberger et al. [32].

During training, multiple glyph images x_t with varying degrees of added noise are fed to the U-Net which in turn predicts the added noise levels based on

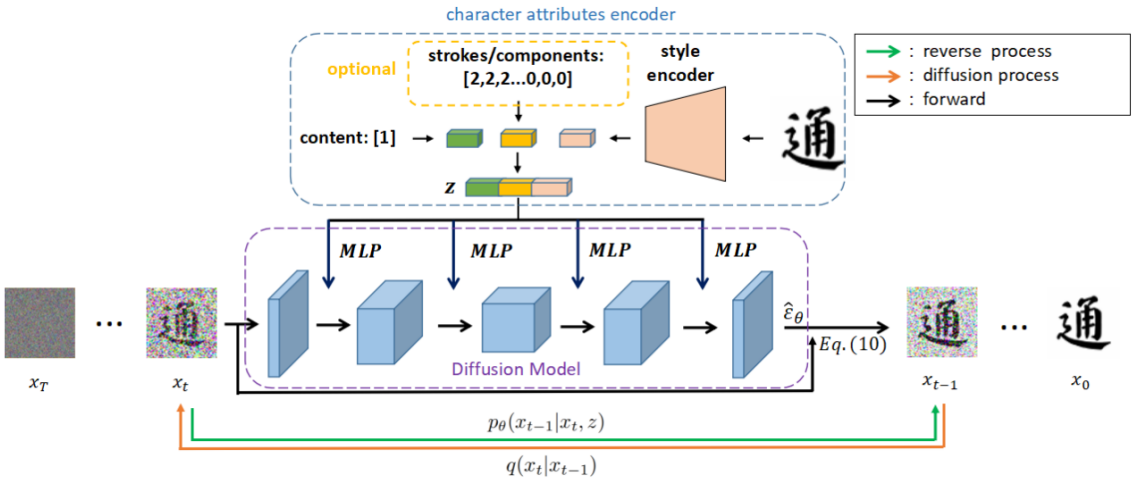

Figure 3.3: Diff-Font overview, taken from He et al. [31].

(x_t, t, z), leading to a simple MSE loss function between the real ϵ and predicted noise $\epsilon_0(x_t, t, z)$. While $t \in [1, 1000]$ denotes the noise timestep (see Figure 3.4), $z = f(c, s)$ is the 1D vector containing the class embedding and encoded style produced by the pre-trained encoder.

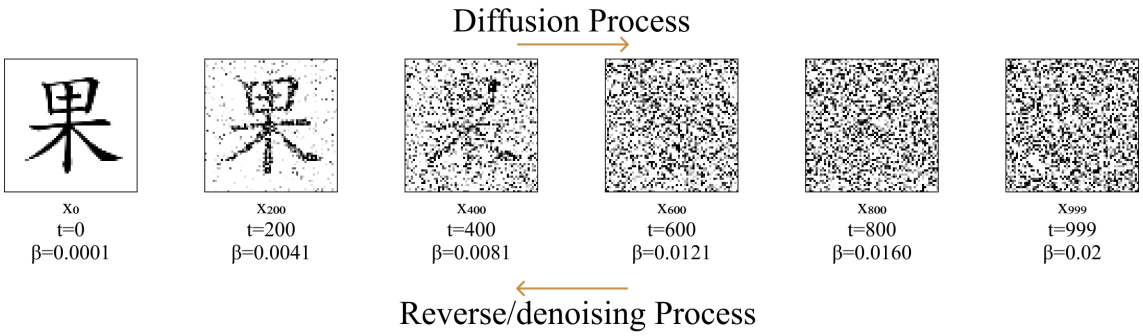

Figure 3.4: Visualization of diffusion and reverse processes. β essentially indicates the amount of noise to be added to the original image. As the timestep increases, β increases in a linear fashion.

At inference time, a glyph with the desired style is given to the style encoder, producing s, and a 64×64 Gaussian noise image x_{1000} is initialized. The so-called reverse/denoising process begins by feeding forward x_{1000} to the diffusion network alongside the newly produced s, $t = 1000$, and the desired glyph class c to which the style will be transferred. The network estimates x_{999} which is then fed back to it repeatedly, completing the denoising process in a total of 1000 steps until a glyph x_0 within the desired class and style is achieved.

GlyphNet, unlike the Diff-Font model, has the capability to generate multiple glyphs simultaneously. In the case of Diff-Font, the process mentioned earlier must be repeated 26 times in order to generate all the glyphs comprising a particular font. This limitation can result in slower generation times, as discussed

in more detail in Section 4.3. Additionally, a fundamental distinction lies in the fact that GlyphNet is regarded as a multi-shot approach, while Diff-Font follows a one-shot approach where a single glyph is provided to generate the entire font. In theory, GlyphNet has the ability to combine design elements from multiple input glyphs when conceiving a new font, whereas Diff-Font must extract the style essence solely from a single input.

3.3 Datasets

In order to train generative models and systematically evaluate the performance of the proposed approach and the quality of its outputs, a large and varied enough dataset was required. These requirements led to the use of the Capitals64 dataset collected by Azadi et al. [5] which includes 12K grayscale raster fonts with varying serifs, boldness, and textures, each composed of 26 ASCII uppercase letters framed within 64x64 canvases. The dataset is divided into train, test, and validation splits in a ratio of 10:1:1. A small data subset is shown in Figure 3.5.

Given that all symbols were fit to the same dimensions, some glyphs (i.e. W, M, T) found in the original fonts were deformed. Therefore, any model trained with this dataset is not equipped to learn features such as relative heights and sidebearings.

Figure 3.5: Train samples from *Capitals64* dataset.

Later on, a second dataset was added to the data pool to assess whether training

data size significantly impacted the model performance. This was accomplished using fonts compiled by Parente et al. [52] from Google Fonts. This is a smaller dataset composed of 6k fonts but includes much more variation of font styles as seen in Figure 3.6.

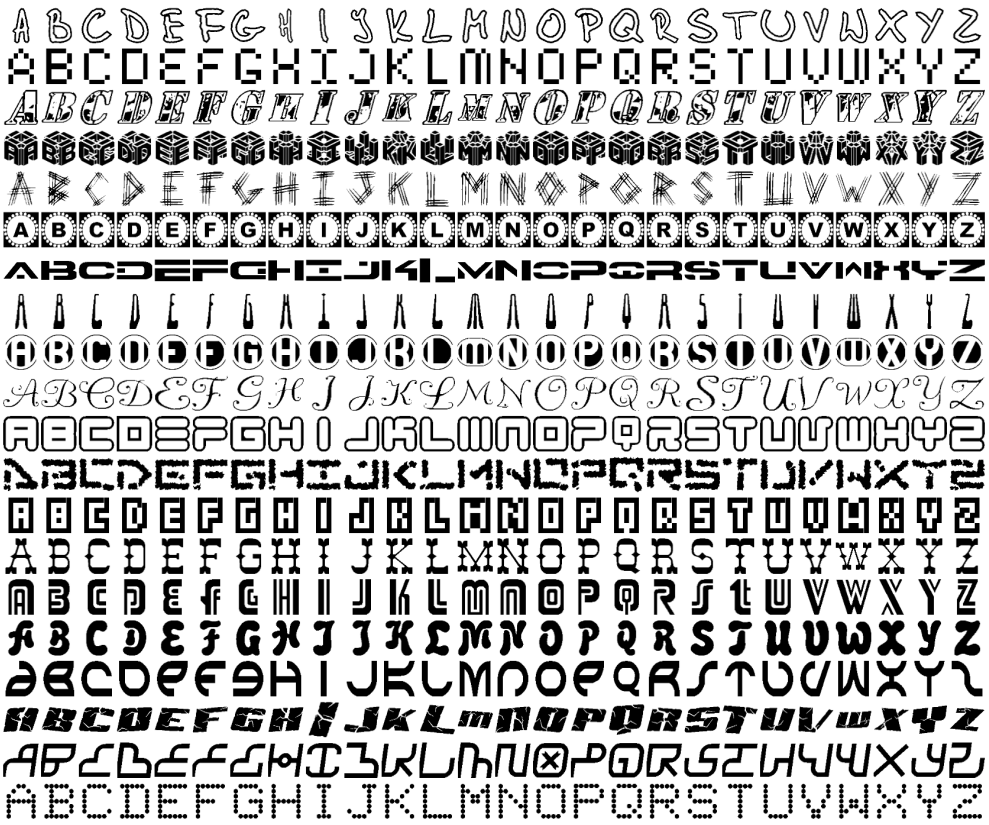

Figure 3.6: Train samples from the dataset compiled by Parente et al. [52].

3.4 Evaluation Metrics

In order to compare the performance of different models across the conducted experiments, FID [33], MSE and SSIM [54] were chosen as quantitative evaluation metrics. While SSIM is a metric that measures similarities between two images, according to luminance, contrast, and structure, MSE computes the mean square errors of pixel values and FID measures the difference between generated and real images in a distribution-wise manner with the aid of *InceptionNet* [34] (as previously studied at the end of Section 2.2). MSE and SSIM are computationally less complex, while FID offers quantitative evaluation closer to the visual perception of humans [55]. These are some of the metrics commonly employed in computer vision tasks and, more specifically, the ones used in the evaluation of current state-of-the-art approaches to font generation [5, 7, 31, 35]. Due to the stochastic nature of the training process, each model configuration was trained in up to 15 distinct RNG seeds and the mean values of each metric were computed on the test data.

Chapter 4

Experiments and Results

This chapter showcases the visual results and quantitative analysis performed on the models discussed in the previous chapter, designed to support our automated font generation approach. The numerical analysis serves as a benchmark for comparing the subsequent modifications we propose for these state-of-the-art generative methods, aiming to enhance the quality of their outputs.

We begin by defining experimental setups and benchmarking the original Glyph Network in light of multiple quantitative metrics in Section 4.1. This thorough analysis leads to the identification of some shortcomings in the original model as well as key insights into how the inputs affect the overall performance. In Sections 4.1 and 4.2, we propose core changes to the loss function of the network in order to overcome these weaknesses. These changes are duly tested alongside further experiments on the network inputs. Finally in Section 4.3, a novel approach to the generation of Roman characters based on Diff-Font is presented in hopes of circumventing the downsides of GAN-based approaches altogether.

4.1 Glyph Network Analysis and Attention Based Loss

The original Glyph Network can receive up to 26 arbitrary characters as input to generate a single given font, meaning that it empowers the user to handpick the initial seedings that will kickstart the font creation process. Given the absence of objective analysis in the original paper, we conducted a comprehensive study to assess the performance of the network when systematically different inputs were provided. We evaluated the results based on metrics such as MSE, SSIM, and FID to obtain quantitative measures of its performance.

To account for the inherent variability caused by the stochastic nature of the network initialization and training process, a total of 15 models were trained using the Capitals64 training data. Each model was trained using only 3 randomly selected input glyphs per font. Subsequently, these trained models underwent 26 separate tests, with each test utilizing a single fixed glyph as input. To ensure randomness had no impact on the final results, multiple models were trained using different random number generator (RNG) seeds, specifically ranging from 0

to 14.

Parameter	Value
Epochs	300
Batch Size	64
Training Pool	9121 fonts
Validation Pool	1472 fonts
Test Pool	1559 fonts
Learning Rate	0.0002
L_1 weight λ_1	100
Optimizer	Adam
LR Decay	Linear in last 100 epochs
Adam Momentum Decay β_1	0.5
Leaky ReLu Negative Slope	0.2

Table 4.1: Hyper-parameter values used during training in all experiments with Glyph Network.

By assessing how each isolated glyph impacted the overall performance of the network, this experiment enabled the empirical search for the most informative letters in font generation contexts thus providing valuable insights to users of such generative systems. It also allowed the establishment of a performance baseline for future experiments. To accomplish this evaluation we set the hyper-parameters to the values defined in Table 4.1 throughout all the following experiments.

After the training process, each model was tested on Capitals64 test data and its performance was evaluated using MSE, SSIM, and FID scores. The quantitative results were then grouped by input glyph. Results depicted in Figure 4.1 reveal that, when fed to the network individually, K, R and H contribute the most to the reconstruction of the original dataset while I, J and T are the least informative. This trend remained equal across FID, SSIM and MSE metrics. The results are in line with visual human perception, since, the capital letters I, J, and T often lack round and diagonal strokes which would make it difficult to guess the appearance of other glyphs containing such features. The letter ranking experiment highlights the significant influence of input selection on the quality of results obtained from both one-shot and multi-shot approaches like GlyphNet. However, it is not sufficient to assess the optimal subset of $N > 1$ input glyphs. It is theoretically possible for the combination of two suboptimal glyphs to contain more style information than a subset consisting of the best-performing glyphs (namely, K and H). This concern is subsequently addressed in further experiments.

Even though this generative approach reaches better numerical results when compared to other state-of-the-art systems, the visual outputs often come with defects that hinder its integration in our final proposed approach. As illustrated in Figure 4.5, the network is capable of capturing the overall style present in the original data but struggles with highly ornamented fonts. Additionally, it does not accurately recreate intricate details, leading to jagged and noisy outcomes instead of what would otherwise be straight lines. These shortcomings are evidenced by the

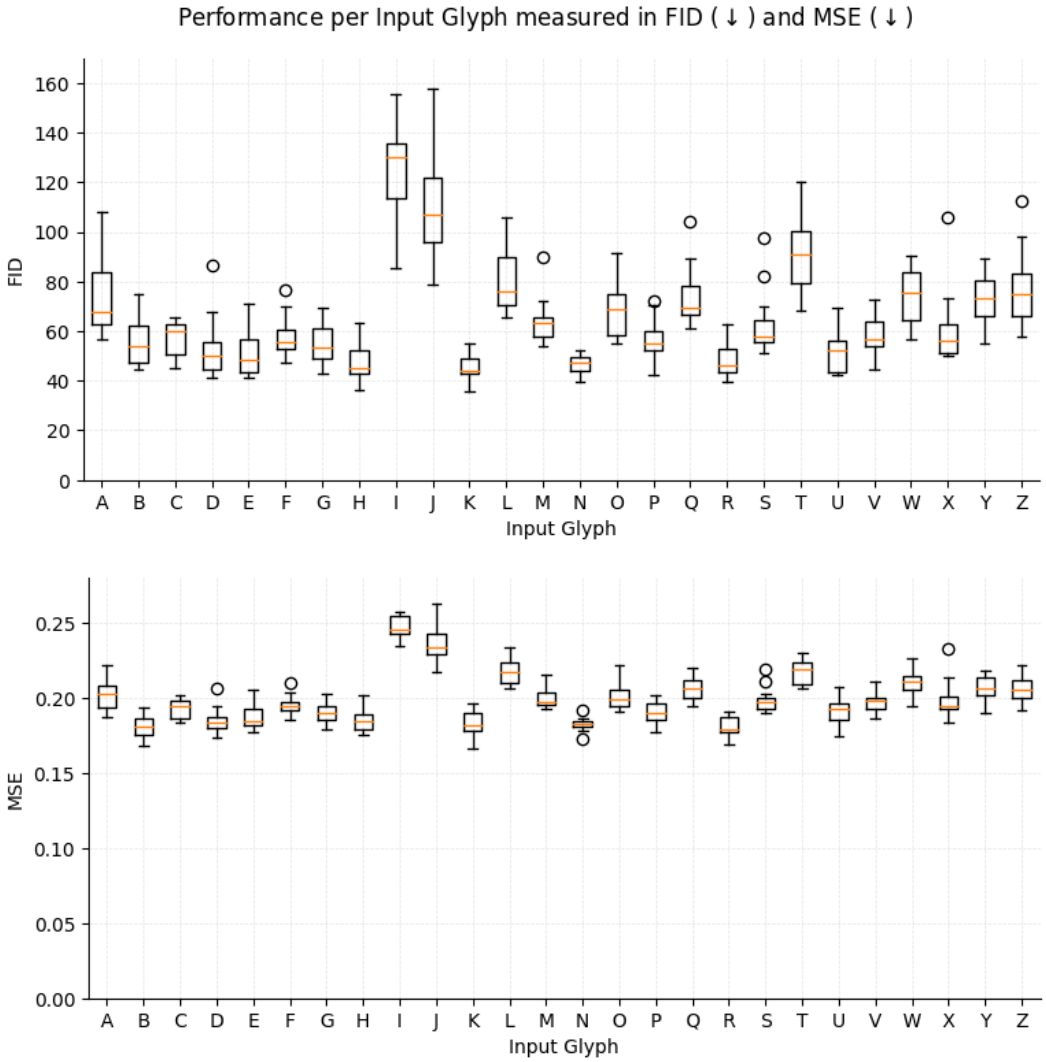

Figure 4.1: Letter ranking with the original loss setup based on FID (top) and MSE (bottom).

blue annotations in Figure 4.5, leading to the conception of the following novel contribution.

To guide the training process to retain the particularities of each target typeface, the L1 loss component was modified in order to accommodate attention maps of the same size as the model inputs (26x64x64). For each training example, a unique weight mask was built in order to attribute more importance to regions of the input tensor that contained distinctive font features. The final L1 loss component of the generator loss function is obtained by applying the dot-product of the attention maps with both network outputs and respective targets:

$$\mathcal{L}(G_1) = \lambda_1 \mathcal{L}_{L_1}(y_1 \cdot M, G_1(x_1) \cdot M), \tag{4.1}$$

where $G_1(x_1)$ is the network output given sample x_1 with target y_1 and M is the attention map. These three parameters are matrices of size (26x64x64).

We explored the first version of these masks by computing the pixel-wise difference between a font devoid of ornaments (Code New Roman) and the target font.

In a second version, these were obtained by applying the Zhang-Suen thinning algorithm [45] to each glyph, in order to bridge the flaws found in some glyph masks where they were very similar to the default font (i.e. resulting in an unusable attention mask). The Zhang-Suen thinning algorithm is an algorithm used to thin 2d objects found in binary images. When applied to glyphs, this thinning process leads to a close resemblance of glyph skeletons which can be further processed to identify joints and terminals. A few examples of these two processes are shown in Figure 4.2. In the following experiments, the three loss configurations will be referred to as $L1_{original}$, $L1_{attention}$ and $L1_{skeleton}$, respectively.

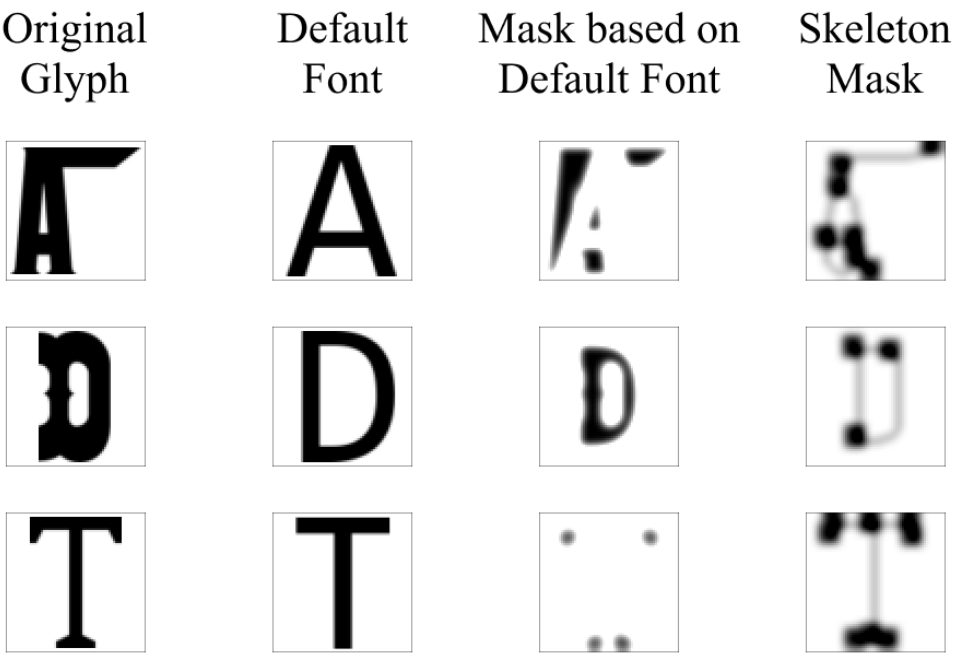

Figure 4.2: Examples of masks used for the computation of the L1 loss term. Original glyphs (1st column), default font with minimal ornaments (2nd column), the mask computed using the pixel-wise difference between glyphs of original and default font (3rd column) and mask obtained with the Zhang-Suen thinning algorithm (4th column).

The previous single glyph experiment was then repeated two more separate times with the new alternative loss functions $L1_{attention}$ and $L1_{skeleton}$, leading to the results shown in Figures 4.3 and 4.4. By repeating the experiment in which a single glyph is visible at a time, we were able to evaluate the effectiveness of these novel loss approaches in handling diverse glyph structures. Furthermore, we compared their performance with the baseline GlyphNet setup, allowing us to assess their robustness.

The trends observed with the original loss remained equal in the new loss setups and across all three metrics. Moreover, when the training loss is directly tied to glyph-specific ornaments, *GlyphNet* achieves slightly better performance. This indicates our novel attempt at guiding glyph generation through attention masks can lead to better quantitative results. Moreover, the performance variance mea-

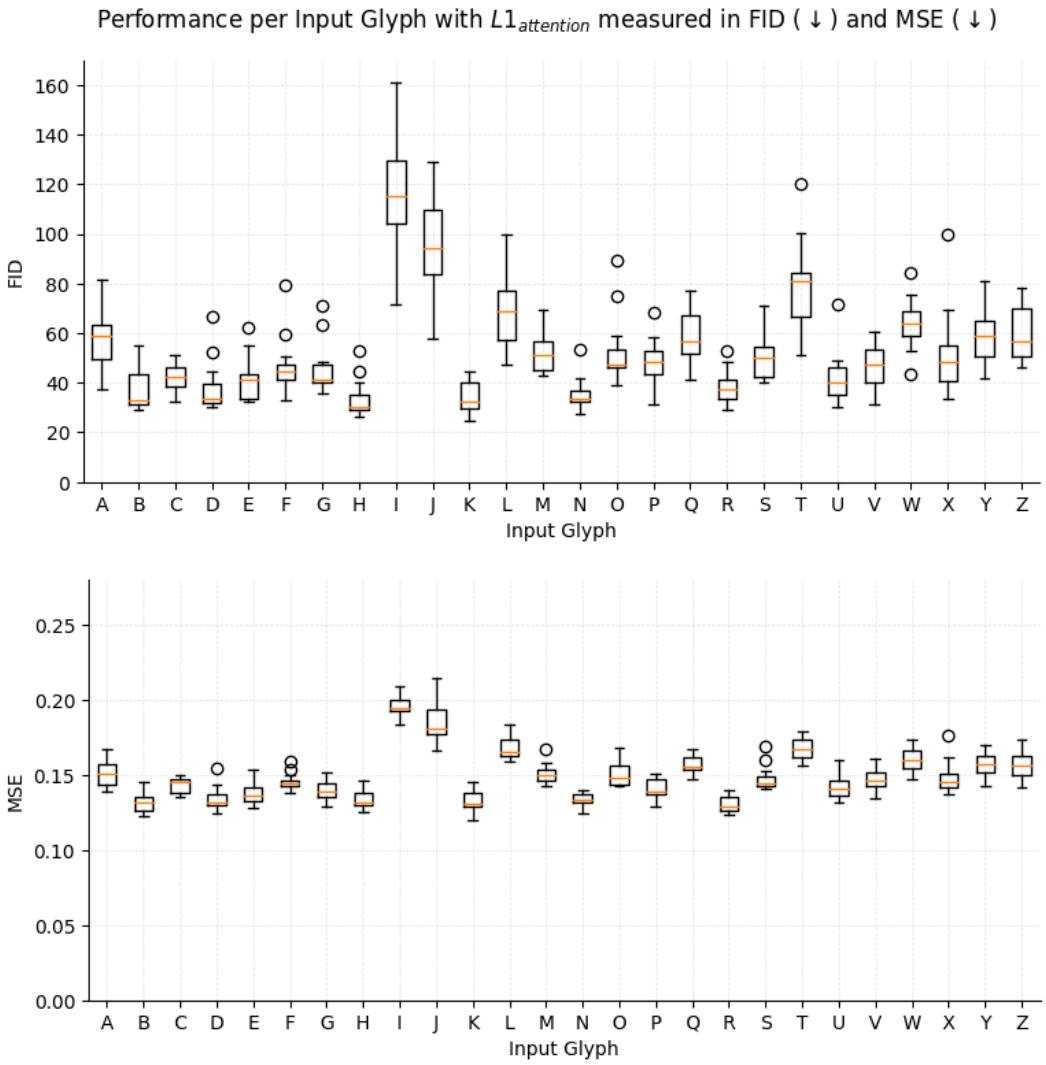

Figure 4.3: Letter ranking with the $L1_{attention}$ loss setup based on FID (top) and MSE (bottom).

sured over multiple models remained equal, specifically for glyphs I, J, and T.

However, comparative results shown in Figure 4.4 (b) between $L1_{attention}$ and $L1_{skeleton}$ models are not conclusive since performance differences are inconsistent across different input glyphs. For example, given that the capital letter O does not contain joints or terminals in most typefaces, the attention mask computed using the Zhang-Suen Algorithm is not as effective, hence the decrease in performance. For that reason, the remaining experiments presented in this dissertation were conducted using the $L1_{attention}$ loss function in place of the original one.

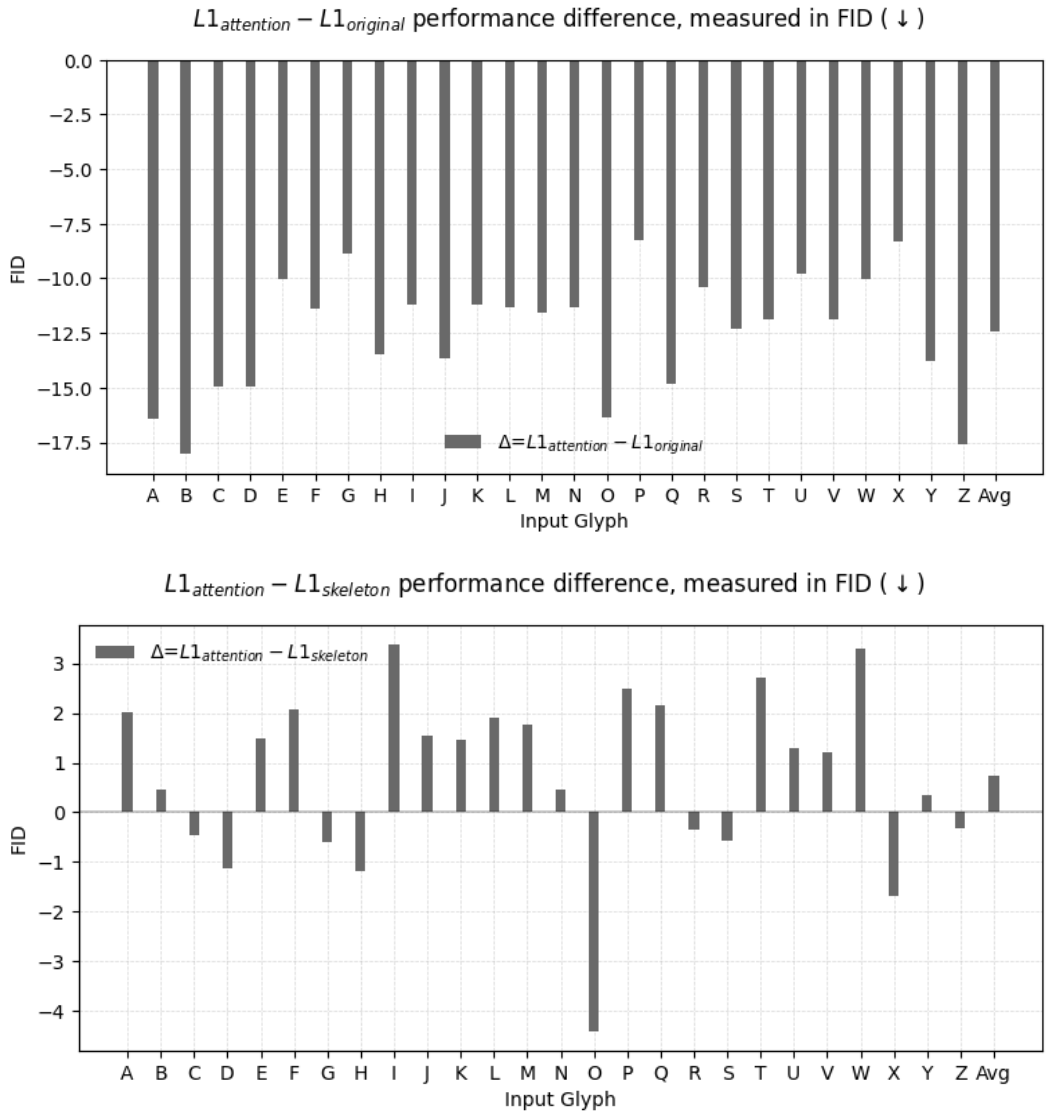

Figure 4.4: Letter ranking with the original loss setup based on FID (top) and MSE (bottom).

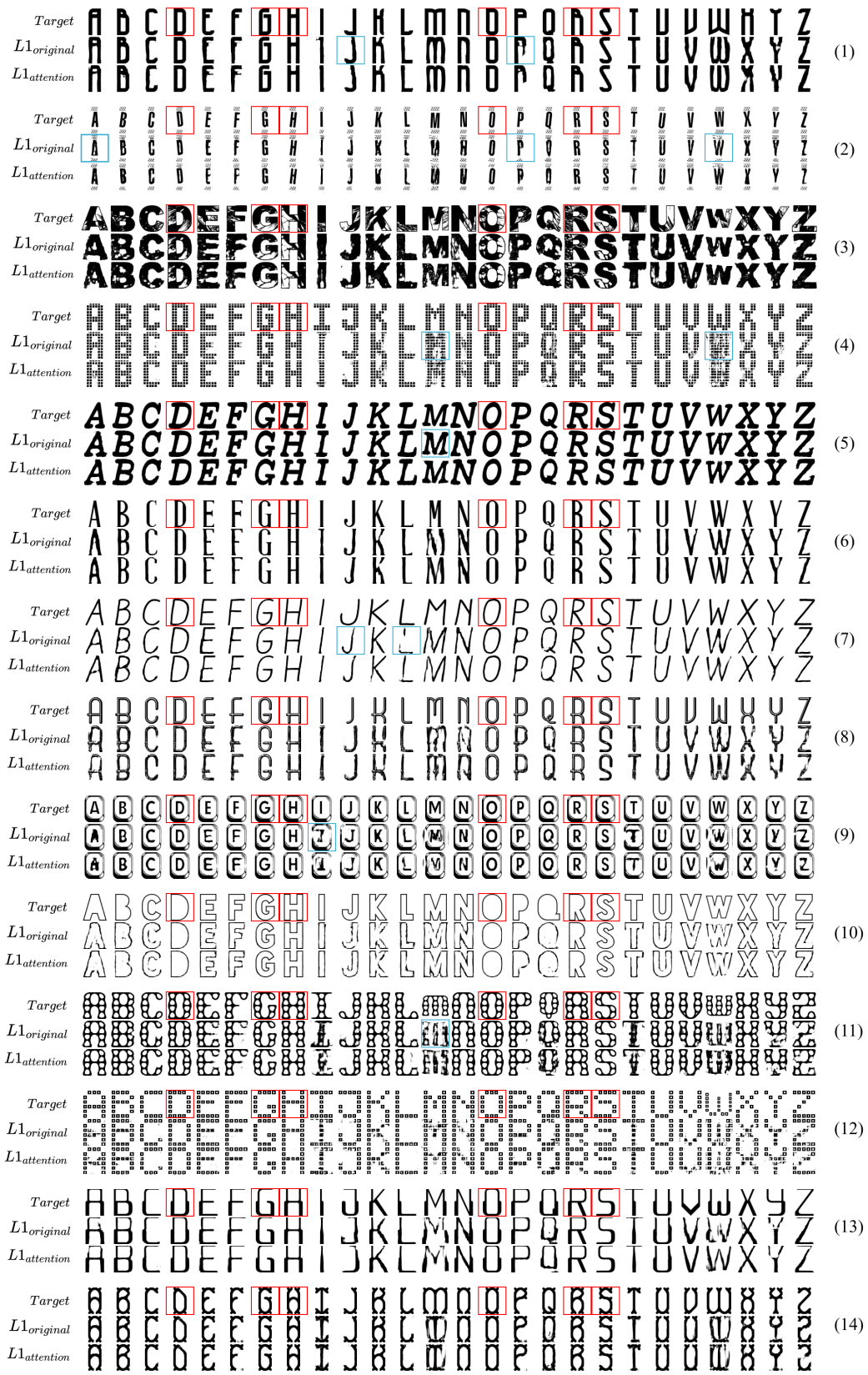

Figure 4.5: Examples of GlyphNet outputs given the subset (D,G,H,O,S,R) as input for test samples. Each row is composed of the target font at the top with inputs annotated in red, followed by the generated outputs. Some undesirable artifacts found on inferred glyphs are marked in blue squares.

4.2 Multi-shot Experiments

After studying how each individual glyph contributed to font inference, a subset of the 6 best-performing glyphs (B,D,K,H,N,R) was stacked to compare against a subset that a typeface designer typically starts the designing process with for a novel typeface (D,G,H,O,S,R) [56].

In this experiment, 15 different runs for the two groups of 6 letters were collected, each composed of model instances trained and tested solely on one of the input subsets. $L1_{attention}$ loss was used while all other setup elements remained unaltered. The decision to adopt $L1_{attention}$ loss in the following experiments was based on the increased performance measured against $L1_{original}$ loss. On average, the new loss function obtained 12.5 FID scores lower than the original one. The variation in performance observed with $L1_{skeleton}$ was not enough to continue using it.

Table 4.2: Glyph Network performance metrics per subset for the 15 different random seed runs with $L1_{attention}$.

	$\{K, H, B, N, R, D\}$			$\{H, O, D, S, G, R\}$		
	MSE(↓)	FID(↓)	SSIM(↑)	MSE(↓)	FID(↓)	SSIM(↑)
Mean	0.0611	7.4264	0.7283	0.059	6.5348	0.7372
StD	0.0027	1.5386	0.0117	0.0002	0.0701	0.0007

Despite subset 1 being composed of the most informative letters found from the first experiment, it showed less precision and lower generative performance when compared to subset 2, as shown in Table 4.2. The differences in performance measured with MSE between these two subsets are statistically significant according to the Wilcoxon Signed-rank test with a 95% confidence level ($p - value = 0.03125$).

The performance of the single glyph experiment significantly improves when the number of inputs is increased from 1 to 6 and they are constant across all training fonts, as observed when comparing the evaluation metrics to the previous multi-shot setup. These results are further reinforced by the preliminary study conducted in the original paper, which also utilized multiple input glyphs to obtain their visual outputs. Furthermore, since the original Glyph Network model can accommodate up to 26 input glyphs to generate a given font, it is rather relevant to observe how increasing the number of input glyphs impacts the overall performance of the model. For that reason, in a third experiment, the network was trained and tested on increasingly bigger input sizes. Starting with the 3 best letters found in the Letter Ranking experiment and up to the top 9, the network was trained using $L1_{attention}$ loss up to 300 epochs. Its performance was measured with MSE, FID and SSIM metrics. Results computed across all three metrics, de-

picted in Figure 4.6, indicate the model performance increases in a linear fashion the more initial examples are given to it.

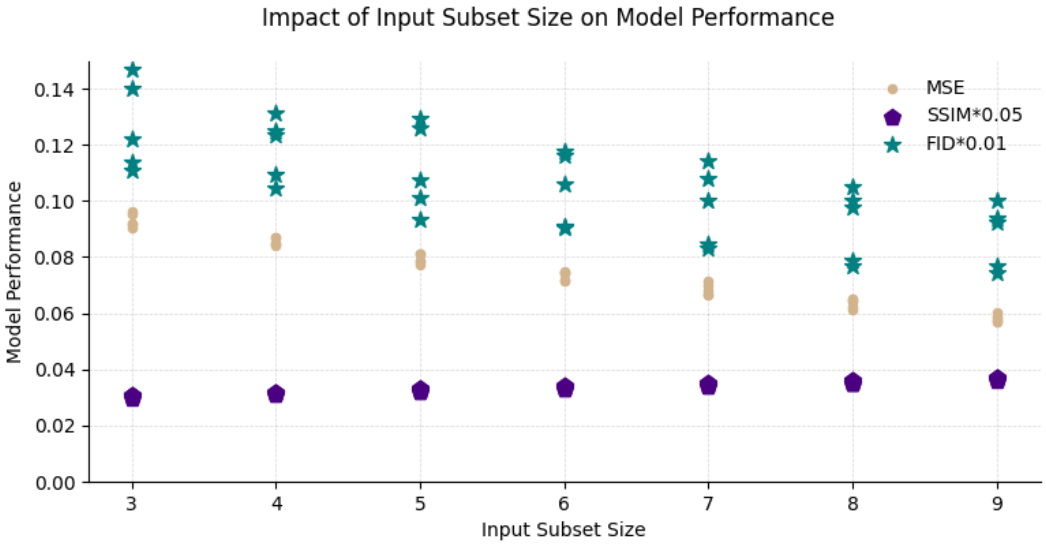

Figure 4.6: Performance evolution over increasingly larger input sizes. For each size, 15 distinct random seeds were used for network initialization. Better quality is indicated by lower MSE and FID, as well as higher SSIM values.

Figure 4.7 visually shows how this experiment was conducted. The first row of each three depicted fonts is composed of the expected targets while the following rows are produced by model instances trained on 3 to 9 fixed input glyphs.

Although one might be tempted to increase the input size even further to obtain fonts with more quality, this strategy clashes with the initial goal of the presented system to aid typeface designers in the creation process of new fonts. If the user has already built a sizable collection of examples on its own, using a generative model to create the remaining characters, which in most cases need to be cleaned and revised, might be more time-consuming than assembling them by hand.

Figure 4.5 shows a varied sample of fonts generated using the original loss function and our novel attention-based approach. Even though the proposed technique rectifies some of the imperfections annotated with blue markers and the quantitative analysis shows objective improvements, it is still not enough to obtain usable glyphs of the most complex fonts found in our dataset pool (see examples 4, 9, 11).

In a last attempt to improve the performance of the network, we expanded the training dataset with fonts from the second dataset presented in section 3.3. To assess whether or not this increase in training data would affect the output quality, a combined test pool was assembled by joining both datasets' test splits. A group of 15 models was trained and tested on the original dataset, while another group was subjected to this new data arrangement, leading to the results of Table 4.3. In spite of the 50% increase in training data, overall the performance of Glyph Network remained unaltered across all metrics.

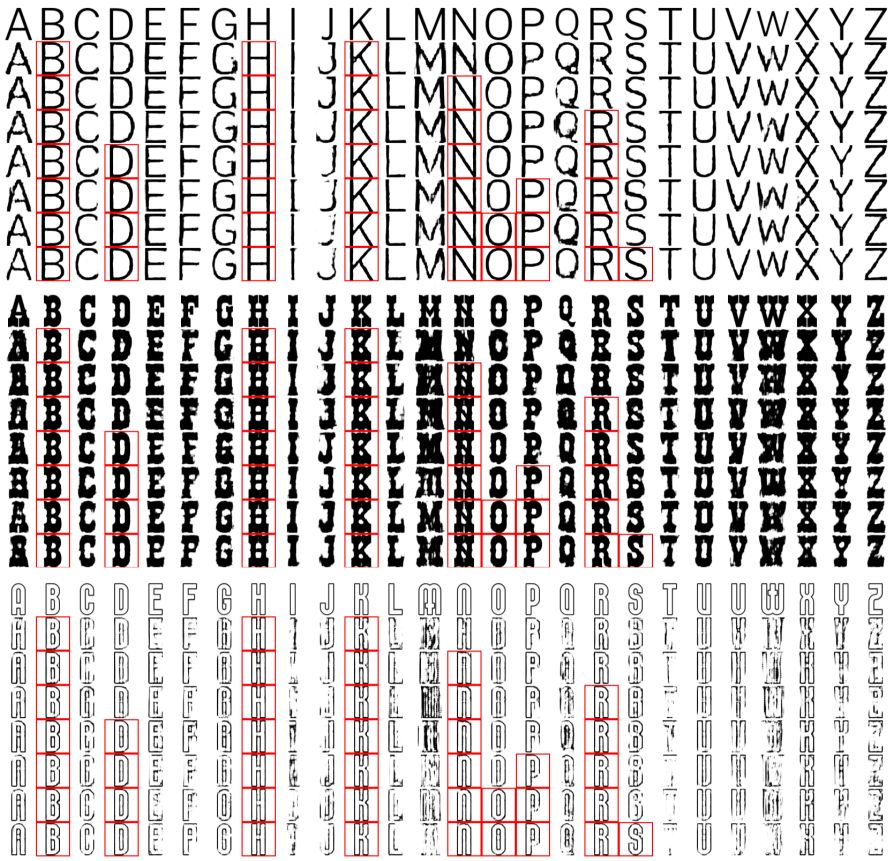

Figure 4.7: Visual examples drawn from the input subset size experiment where increasingly bigger input subsets were given to the Glyph Network. The target of each font is found in the top row while outputs of models with varying input sizes are presented in subsequent rows. Inputs are annotated in red.

Table 4.3: Effect of training data size on Glyph Network performance according to FID, MSE and SSIM

	Original Dataset			Combined Dataset		
	MSE(↓)	FID(↓)	SSIM(↑)	MSE(↓)	FID(↓)	SSIM(↑)
Mean	0.0589	6.4835	0.7359	0.0603	7.1457	0.7276
StD	0.0009	0.153	0.0048	0.0007	0.2367	0.0045

4.3 Diffusion Model Approach

Given that the modifications proposed to the Glyph Network and posterior dataset augmentation were not enough to completely suppress the noise and imperfections found in the generated outputs, a new generative approach was designed based on Diff-Font [31]. This choice was influenced by the observation of recent advancements in diffusion models, which demonstrate the ability to produce authentic images by considering contextual input. These cutting-edge diffusion models have positioned themselves at the forefront of state-of-the-art techniques in visual generative tasks.

The original Diff-Font model is able to generate a given Chinese/Korean character based on another symbol's style features, which are extracted using a pre-trained feature encoder developed in Xie et al. [35]. Here we propose instead the use of a Glyph Network model trained on the character subset (D,G,H,O,S,R) during the previous experiments in place of the former style encoder. Even though this model struggled to reproduce the most complex fonts, it demonstrated its capability to identify key style features in the majority of the test dataset, as concluded in the previous sections. Besides, this substitution makes it so that the Diff-Font inputs are the exact same as the ones in Glyph Network, enabling direct comparisons between these two approaches.

Throughout the training process of this approach, the original hyper-parameters remained unchanged (see Table 4.4), with the exception of the modification made to the already discussed style encoder and the number of ResNet blocks [53] employed in each UNet resolution. The count was reduced from 3 to just 2 due to the limited video memory available during training using a single RTX 3080ti GPU.

Parameter	Value
Batch size	24
Training pool	9121 fonts
Validation pool	1472 fonts
Test pool	1559 fonts
Nº of Res. blocks per resolution	2
Channel multiplier	[1,2,3,4]
Attention resolution	[40,20,10]
Diffusion steps (max t)	1000
Noise schedule	Linear
Training iterations	400k
Learning rate	1e-4
Optimizer	Adam w/o weight decay
Loss	MSE

Table 4.4: Hyper-parameter values used during the training of the diffusion model.

Through the training and subsequent testing of this diffusion approach, we achieved scores of 0.13, 0.52, and 16.5 in MSE, SSIM, and FID metrics, respectively. These

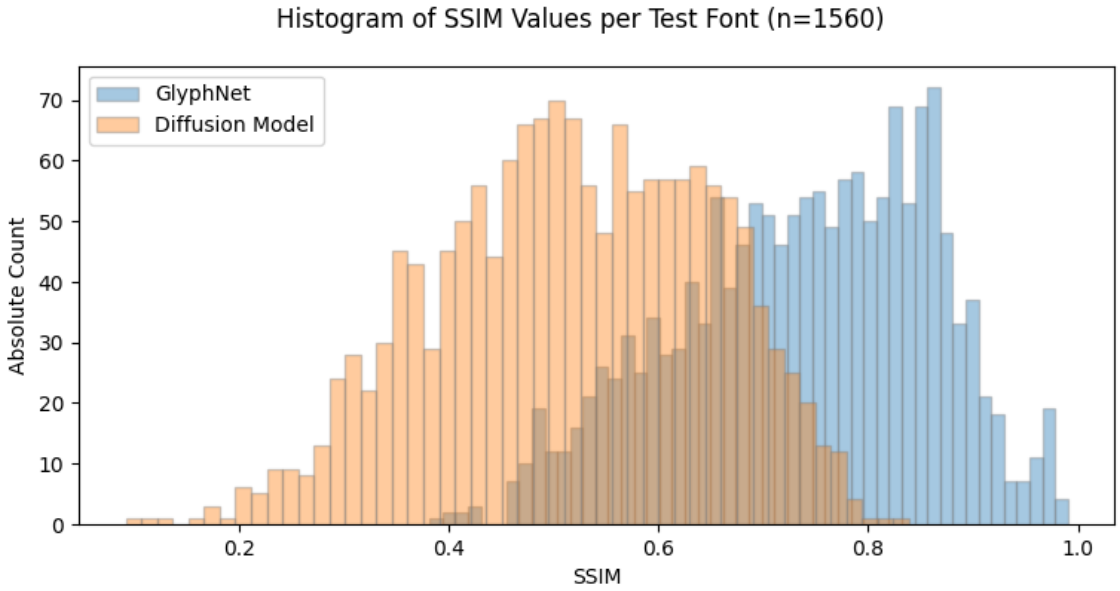

Figure 4.8: Difference in SSIM values achieved by GlyphNet and Diffusion Model. The higher the SSIM, the closer are the artificial fonts to the original ones.

results indicate that this new approach performs worse than GlyphNet overall when using the same inputs. This conclusion is also supported by the SSIM histogram of Figure 4.8 where GlyphNet achieves scores much closer to 1 while the diffusion model is less consistent across different fonts. Additionally, visual examples displayed in Figure 4.10 demonstrate its struggle to capture and reproduce the styles of ornamental fonts, such as (4), (9), and (11). However, the approach excels in eliminating the issue of noise and imperfections when generating simpler fonts like (1), (3), and (7). In these cases, the diffusion model produces glyphs with perfectly straight lines and well-defined edges, in contrast to the Glyph Network's generated glyphs, making it particularly well-suited for users who later intend to vectorize their generated work. The absence of noise and irregularities on the glyphs produced by our diffusion approach is even more noticeable in Figure 4.9. More examples of generated fonts are available in Appendix B.

The primary drawback of utilizing this approach lies in the substantial inference time needed to generate a single glyph. Since each generated image undergoes multiple denoising iterations to produce a single glyph, the time complexity during inference for this approach is significantly higher compared to the Glyph Network. Consequently, we were only able to train and test a single model instance using this approach, as obtaining a small population of trained models would take several months. The time required to train and test the models covered in this chapter is showcased in Table 4.5.

On the other hand, as the generation of new glyphs is governed by the style features encoded within a 1D float vector, it presents an opportunity for interpolating between any two distinct fonts. This interpolation allows for the creation of a new, third font that possesses a customizable degree of resemblance, blending characteristics from the original parent fonts. Figure 4.9 illustrates two instances

Model	Training Time	Inference Time	Memory Footprint
Glyph Network	12.1 hours	$<$ 0.1 secs	4.95 GB
Diffusion Model	25.4 hours	650 secs	12.3 GB

Table 4.5: Time and memory required to train each approach using the setups of tables 4.1 and 4.4. The inference time refers to the time required to generate a single font with 26 characters.

of font interpolation, displaying different levels of blending. Two distinct fonts were given to the style extractor using the input subset (D,G,H,O,S,R) and then those styles were linearly interpolated according to a range of blending weights. The derivatives of this interpolation process were then fed to the UNnet, producing the outputs shown in Figure 4.9. Since the Glyph Network was not originally meant to act as a standalone style encoder, the incapability of this system to reproduce complex styling and subsequently lower output quality can be pointed to this incompatibility. To enhance the overall smoothness of these interpolations, an alternative approach could involve employing the training process of a Variational Autoencoder on the style extractor, rather than relying on a pre-trained GAN as executed in this project.

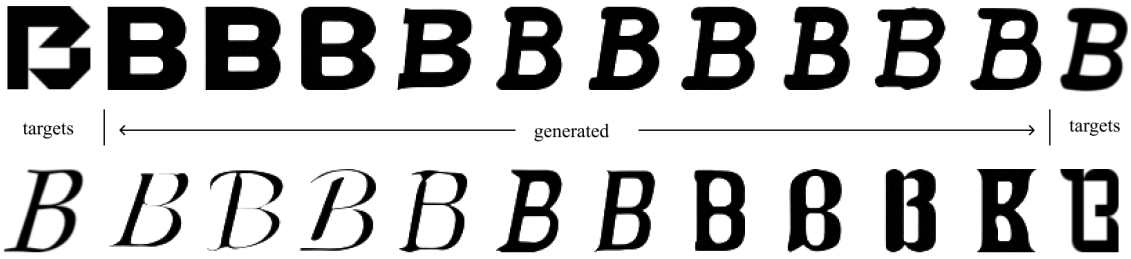

Figure 4.9: Glyph Interpolation using the proposed diffusion model.

In summary, Glyph Network continues to outperform our diffusion method in generating raster fonts from multiple context glyphs, achieving half the MSE score and double values in FID in identical scenarios. Through multi-shot experiments we found six to be the optimal number of input glyphs to be provided, balancing an amount that produces better performance values than other state-of-the-art systems while still maintaining the amount of required work from type designers relatively small. Although the search space for the best combination of six letters is rather vast and computationally unattainable, we propose two distinct subsets that we obtained from studying the most informative glyphs and adhering to design intuition - (B,D,K,H,N,R) and (D,G,H,O,S,R). Moreover, we found no evidence that increasing the amount of data available during training leads to direct benefits in the quantitative performance of our GAN-based model. Even though we achieved scores of 0.059, 0.7, and 6.5 in MSE, SSIM, and FID, respectively, by making improvements to core aspects of the Glyph Network system, it still displays shortcomings associated with its visual outputs. Stroke and edge imperfection issues were partially resolved by adopting a completely new

diffusion approach, which outputs overall cleaner glyphs. Furthermore, there remains untapped potential for further experimentation and improvement of the proposed diffusion network, particularly in terms of refining its architecture and optimizing hyper-parameters. Additionally, the time required to train and infer new glyphs using this approach needs to be shortened in order to allow future experiments to be conducted in a tangible manner.

Figure 4.10: Comparison of outputs produced by the diffusion model and GlyphNet given the subset (D,G,H,O,S,R) as input for test samples. Each row is composed of the target font at the top with inputs annotated in red, followed by the generated outputs.

Chapter 5

Web Application

Having trained and tested multiple generative models capable of producing raster glyphs and fonts with satisfactory quality, we reached a point where the development of a comprehensive software system was viable. This system would embody the entire approach outlined in Section 3 and be presented as a web application. Its purpose would be to serve as a basic proof-of-concept for future platforms designed to assist designers and non-technical users in the creation of innovative typographical fonts. This section details the work conducted in that regard, commencing with the definition of high-level requirements and initial design iterations. It concludes with a demonstration of the application in its final state and an explanation of the choices and modifications implemented.

The primary objectives of this application are to offer a user-friendly method for creating fresh fonts using a selection of arbitrary glyphs provided by the user, utilizing the models exhibited in the preceding Section 4. The user will have control over the generation process, including adjustable settings like the color scheme of the inputs, the output format (raster or vectorized), and the choice of model to be utilized. It should also encapsulate all the modules that compass the initial approach (see Figure 3.1), including the vectorization step carried out by the LIVE framework [42].

The initial application designs depicted in Figure 5.1 were devoid of color as it allowed us to focus on the development of core functionality and usability aspects rather than stylistic choices and also helped to highlight the prototype nature of the app to the end users. Given the sole purpose of the app and the focus on delivering a simple generative workflow, it was made the decision early on to accommodate all functionality in a single webpage divided into three sections:

- Header - where the app identity is succinctly identified and some navigation options are provided such as the dark theme and localization features;
- Canvas - area that mimics a workstation in the sense that it is where all the user-made artifacts are sent to after upload and represented via image cards. This is where the user can preview, remove and correctly label all inputs;

- Control Panel - section underneath the canvas where all the adjustable settings related to the generation procedure reside alongside the generate button to be pressed at the end of the configuration process.

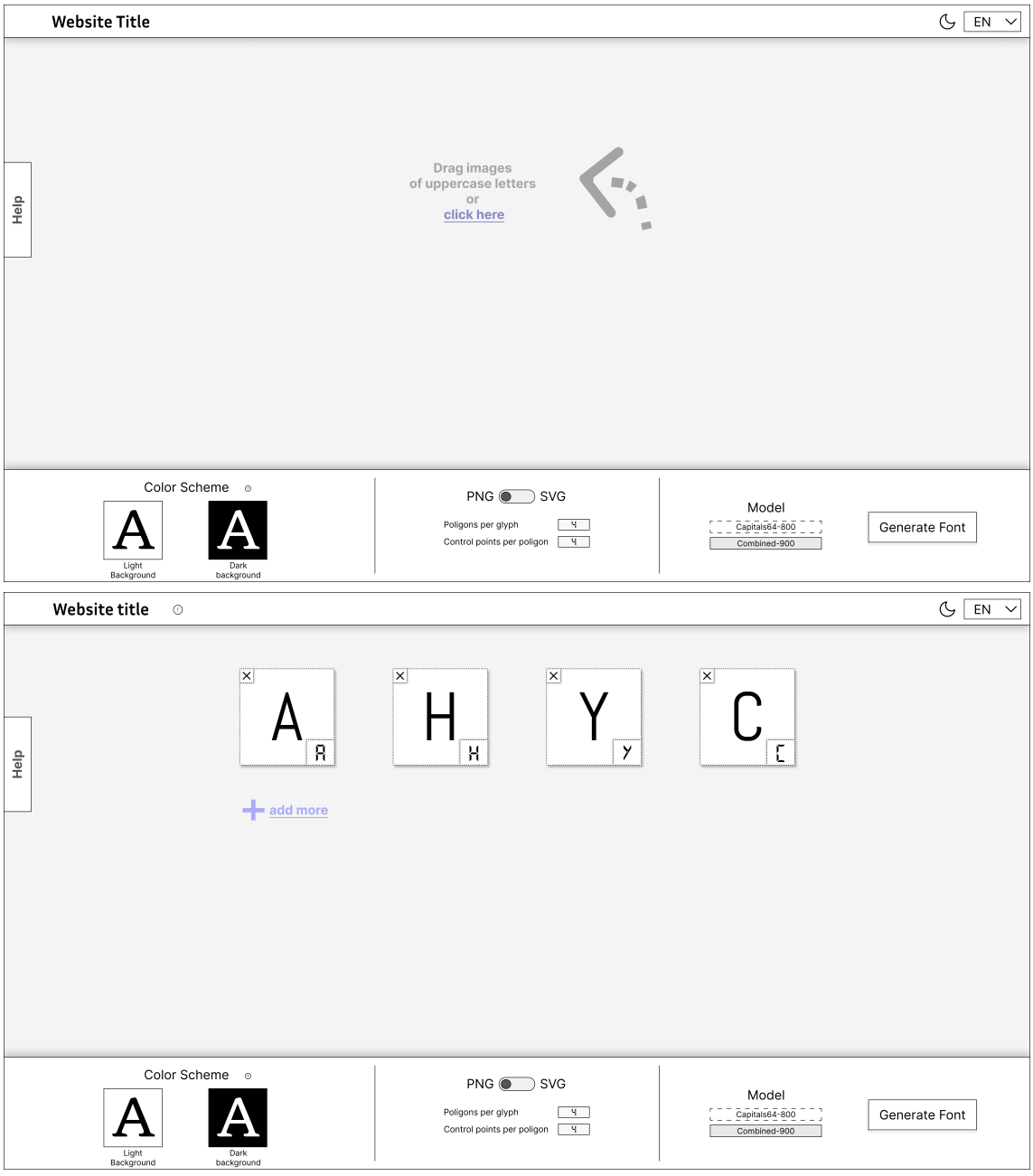

Figure 5.1: Initial mockups of the web application in its default state (top) and when users upload their design inputs (bottom).

When the user clicks on the generate font button, the inputs are uploaded to the server which in turn generates the font and triggers the download of the file on the user's browser. This file is either a single PNG or a Zip containing 26 SVG files depending on the output format chosen prior.

The first improvement made to the app was the addition of a loading animation visible while the font is being generated by the app server as well as an Estimated

Time of Arrival (ETA) indicator in order to clearly transmit that this process requires some time to complete. Other quality-of-life improvements include setting default values for each control panel option to accelerate the workflow, hiding the SVG options when PNG is selected, and the addition of a clear description of what those settings mean as per the explanation given when the LIVE framework was mentioned in Section 2.

The biggest change made to the initial mockups was the communication to the user of which glyphs each model expects as initial seedings. Given that each provided model may be trained in different glyph subsets, it is desired that the user inputs match them to attain optimal output quality. This communication is made possible with the appearance of upload cards in the canvas section corresponding to the expected inputs for the model selected at the time. However, the user is still free to upload any glyph it desires. A short summary of each model is also shown when users hover the mouse over the model list found in the control panel.

Overall, the final application remained faithful to the initial mockups with slight improvements in its usability as showcased in the screenshots of Figure 5.3. The UI was fully developed with React and hosted through NextJS while the backend is supported by a Django server written in Python. When users upload their designs and attempt to generate a new font, the React page initiates an API request to the backend, which proceeds to fetch the desired pre-trained model and generates the outputs that are then sent back in response to the users' browsers. An example of the outputs that users can obtain with the application is shown in Figure 5.2.

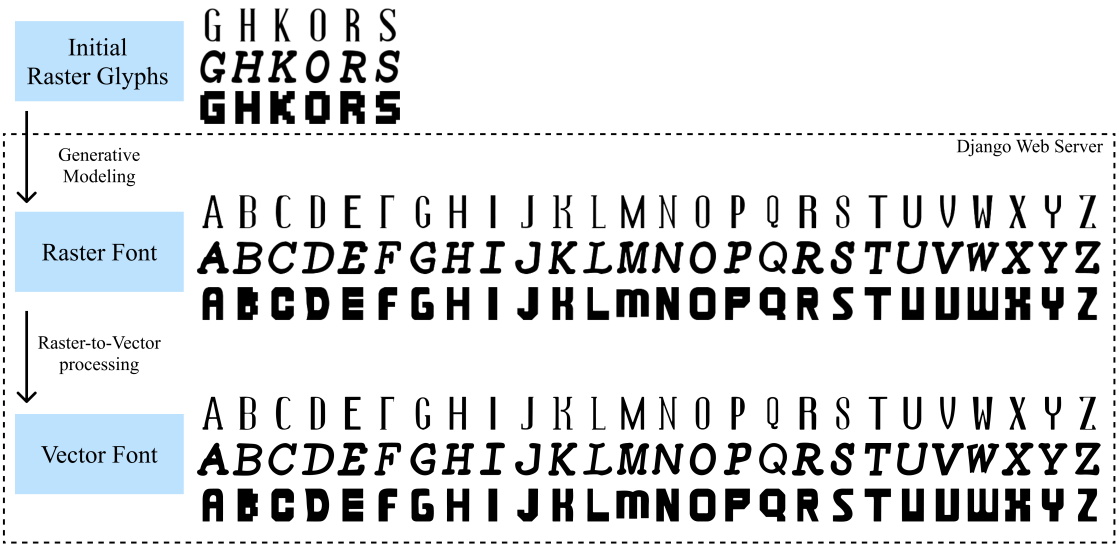

Figure 5.2: Examples of fonts generated using our sequential approach. The model uses the inputs provided by the user to capture the style and generate the remaining glyphs. These glyphs can then be converted to vectorized representations using off-the-shelf libraries.

By adopting this approach, we have successfully achieved our objective of showcasing our contributions to non-technical users via a user-friendly web interface

that enables the swift generation of new fonts within minutes. Moving forward, future development efforts may involve further exploration and optimization of this application to ensure proper scalability for medium to large user bases. Additionally, there is potential to introduce new features, such as font interpolation as discussed in Section 4.3. Moreover, alternative mediums could be explored, including the development of plugins for popular Adobe products (e.g. Photoshop, Illustrator) and native OS applications that would operate entirely on local machines. These expansions would broaden the reach and versatility of our font generation framework.

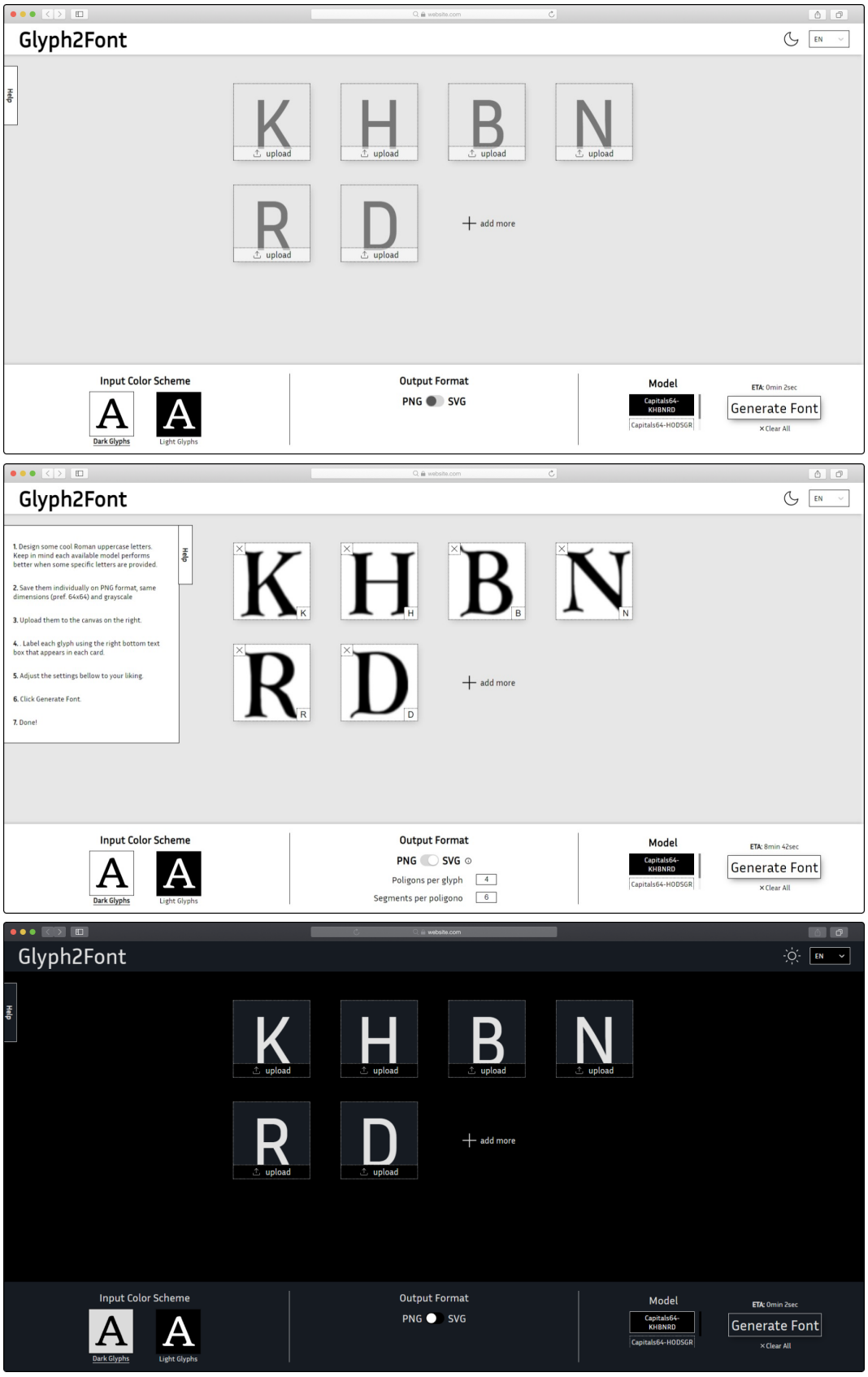

Figure 5.3: Final version of the web application.

Chapter 6

Methodology and Scheduling

Because of the project's investigative nature, we opted for the Waterfall model [57] to carry out all the activities. This approach involved executing each major task in a sequential manner, including initial research, SOTA replication, model development, and testing. By following this methodology, we ensured that each task was initiated only after the completion or significant progress of its preceding counterparts.

Throughout the initial semester, the primary focus was on conducting an extensive survey of the prevailing font generation techniques. Additionally, we explored a wide range of image generation methods that had not yet been applied to address the specific problem at hand as well as potential data sources to be used during future experiments. The initial survey allowed us to pick a few candidate methods that would later compose our own generative approach. This research method is reflected in the schedule outlined in Figure 6.1, which specifies the individual tasks performed during the first semester of the project.

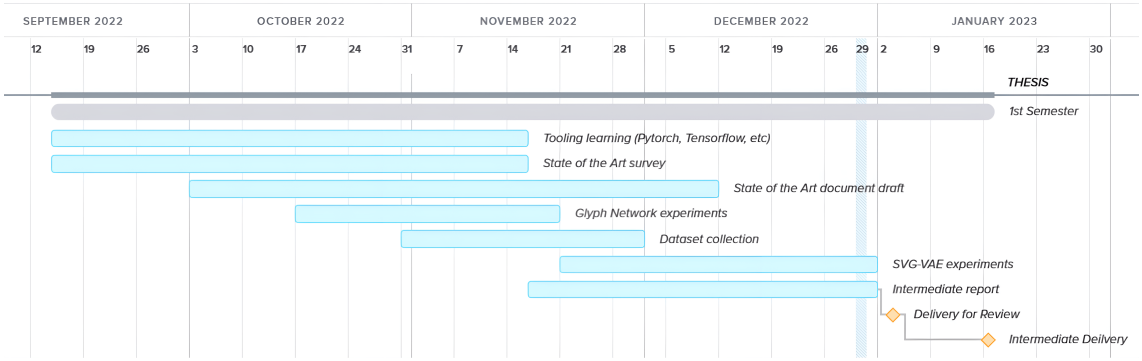

Figure 6.1: Schedule followed during the 1st Semester (September to January).

The initially proposed approach contemplated the development of two parallel models - one based on the Glyph Network and another based on SVG-VAE [36], which aimed at producing fonts entirely in vector format. To accommodate the research and experimentation with these two competing models, an initial schedule was outlined, resulting in the Gantt chart presented in Figure 6.1.

Due to a lack of publicly available datasets and reproducibility issues encoun-

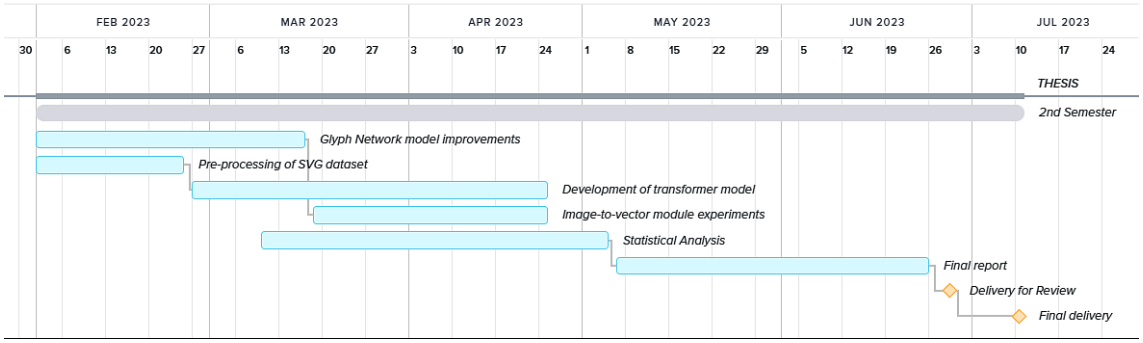

Figure 6.2: Initial Gantt chart outlining the work planned for the 2nd Semester (February to June).

tered during replication efforts, the SVG-VAE model was dropped in favor of a novel diffusion approach that had not yet been published at the time of the initial SOTA survey. This change is also in line with the findings produced with the Glyph Network, as described in Section 4.3, and recent research conducted in the image generation domain. Furthermore, the depth and extent of experiments carried out with the Glyph Network exceeded initial expectations, resulting in the second semester's schedule being predominantly dedicated to the development of this particular model. These experiments resulted in a collection of contributions presented in our paper submission to the 6th European Conference on Artificial Intelligence (ECAI 2023) entitled "Improving Glyph Network Performance through Attention Loss Function and Input Selection: A Quantitative Analysis", which can be found in Appendix A.

Lastly, the project's contributions to the initial problem have paved the way for the creation of a web application, showcased in Section 5, which demonstrates the practical use cases of our complete approach. This supplementary task was integrated into the final weeks of the project, as illustrated in Figure 6.3. Overall all the work was conducted without too much friction and within all initially established deadlines, even though, for instance, it heavily relied on the availability of open-source code and data.

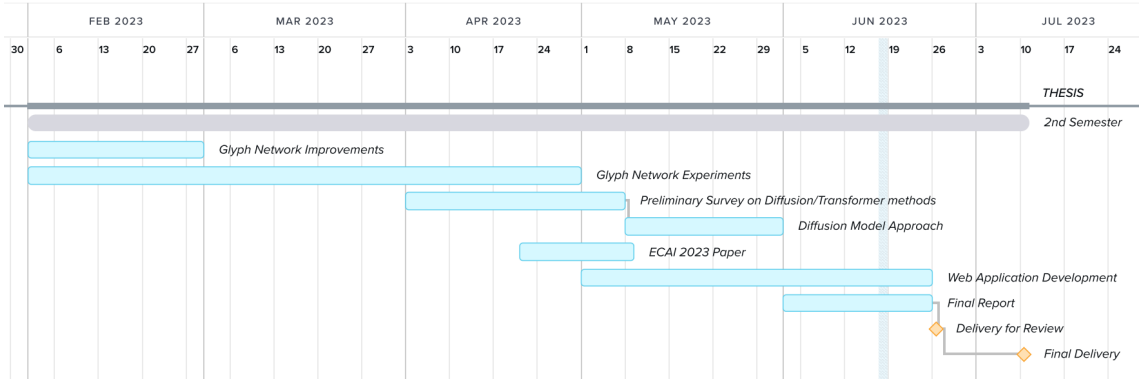

Figure 6.3: Actual schedule followed during the 2nd Semester.

Chapter 7

Conclusion

Designing typographic fonts has proven to be often a complex and time-consuming process, which is an issue that many times restricts the creative possibilities of designers during the creation processes.

With this dissertation, we aimed to contribute by studying and proposing a computational system to hasten the type design process. More specifically, we intended to develop a system that can take one or a few manually designed glyphs as input and automatically generates a whole congruent font, in-style with the input glyphs.

To achieve that, in this dissertation, we began our studies by contextualizing and exposing the problem of font generation and explaining the intricate challenges that follow it when considering the use of generative models as solutions. After that, a comprehensive analysis of the state-of-the-art was presented alongside theoretical explanations of the complex generative models that support each referenced work.

After assessing the strengths and shortcomings of the most prominent state-of-the-art solutions, we propose a unified approach that tackles the issues of font generation and posterior vectorization in separate pipeline stages. By adopting this strategy we were able to conduct an in-depth study of the performance of the Glyph Network and evaluate its behavior when different glyph inputs are provided to it. Furthermore, this study allowed us to modify core characteristics such as the loss function, thus, improving the performance of the Glyph Network even further in the raster domain. In the end, the main contributions are a loss function that provides better results than the state-of-the-art approach, a selection of the best glyphs to produce cohesive fonts and a systematic analysis of the experiments that led to these contributions. The code for the approaches and experiments to generate the results of this dissertation is readily available [1].

After conducting a series of multi-shot experiments, we determined that providing six input glyphs yields optimal results. This balance allows for improved performance compared to other advanced systems, while still keeping the workload for type designers relatively manageable. Although the search space for find-

[1] https://github.com/FranciscoBrilhante/master-thesis

ing the best combination of six letters is vast and computationally challenging, we proposed two distinct subsets based on our analysis of the most informative glyphs and through design intuition: (B, D, K, H, N, R) and (D, G, H, O, S, R). Additionally, our findings indicate that increasing the amount of available training data does not directly benefit the quantitative performance of our GAN-based model. In one-shot scenarios, our enhancements to Glyph Network resulted in a positive average decrease of 12.5 in the FID score and a 0.5 decrease in MSE values. Using the subset (D, G, H, O, S, R), this network achieved scores of 0.059, 0.7, and 6.5 in MSE, SSIM, and FID metrics, respectively.

Not fully satisfied with the visual outcomes attained through the previous approach, we have developed a novel method for producing Roman letters. Our method draws inspiration from the diffusion model proposed in Diff-Font, marking the first utilization of this technique in this particular task. In specific scenarios, our approach yielded comparable results to previous techniques, while also exhibiting the potential to surpass them in terms of overall visual quality perceived across the datasets covered in this dissertation. Using the same subset (D, G, H, O, S, R) and data employed in the previous approach, the diffusion model achieved scores of 0.13, 0.52, and 16.5 in MSE, SSIM, and FID metrics, respectively.

Having obtained models capable of generating a wide variety of designs, we then designed and developed a web application that could showcase key use cases of our proposed approach to the general public as well as employ the abovementioned innovations in real scenarios. This application also serves the purpose of demonstrating the use of the Live framework in the vectorization stage of our approach, thus attaining the global objective of employing generative systems in vector font creation.

To achieve enhanced performance scores, future work is foreseen in refining the architecture and training process of the proposed diffusion network. More specifically, improvements to the style-extracting process could bring the capability of latent space exploration closer to other approaches specialized in that task while attaining better visual results overall. Additionally, it is important to note that this dissertation did not encompass the generation of letters and glyphs from other writing systems. This limitation arose from the absence of publicly available datasets in those domains and the time constraints that hindered data collection efforts. However, in the future, the acquisition of extensive collections of copyright-free fonts could facilitate the improvement of existing models and the development of novel ones. Furthermore, the significance of modern generative techniques for font design lies not only in their conception but also in their successful integration into existing software with real-world applications. Therefore, addressing challenges related to scalability, explainability, and ethics becomes a pressing matter to be tackled in forthcoming research.

References

[1] E. Lupton, *Thinking with Type*. Princeton Architectural Press, 2010.

[2] K. Cheng, *Designing Type*. Laurence King, 2006.

[3] R. Suveeranont and T. Igarashi, "Example-based automatic font generation," in *Smart Graphics* (R. Taylor, P. Boulanger, A. Krüger, and P. Olivier, eds.), (Berlin, Heidelberg), pp. 127–138, Springer Berlin Heidelberg, 2010.

[4] J. Cunha, *Dissertação sobre relações anatómicas entre caracteres de um tipo de letra*. Master thesis, Universidade de Coimbra, September 2013.

[5] S. Azadi, M. Fisher, V. Kim, Z. Wang, E. Shechtman, and T. Darrell, "Multi-content gan for few-shot font style transfer," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7564–7573, 2018.

[6] Y. Tian, "Master chinese calligraphy with connditional adversarial network." `https://kaonashi-tyc.github.io/2017/04/06/zi2zi.html`, 2017. [Accessed 22-Jun-2023].

[7] D. Sun, Q. Zhang, and J. Yang, "Pyramid embedded generative adversarial network for automated font generation," in *2018 24th International Conference on Pattern Recognition (ICPR)*, (Los Alamitos, CA, USA), pp. 976–981, IEEE Computer Society, aug 2018.

[8] D. Lopes, J. a. Correia, and P. Machado, "Adea - evolving glyphs for aiding creativity in typeface design," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, GECCO '20, (New York, NY, USA), p. 97–98, Association for Computing Machinery, 2020.

[9] D. Lopes, J. N. Correia, and P. Machado, "Adea – evolving glyphs for aiding creativity in graphic design "" cdv lab," 2022.

[10] R. Bringhurst, *The Elements of Typographic Style*. Elements of Typographic Style, Hartley & Marks, Publishers, 2005.

[11] W. S. W. Group, "Scalable vector graphics (svg)." `https://www.w3.org/Graphics/SVG/`, 1999.

[12] A. Ng and M. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Advances in Neural Information Processing Systems* (T. Dietterich, S. Becker, and Z. Ghahramani, eds.), vol. 14, MIT Press, 2001.

[13] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014. cite arxiv:1411.1784.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[15] D. Vint, M. Anderson, Y. Yang, C. Ilioudis, G. Di Caterina, and C. Clemente, "Automatic target recognition in low resolution foliage penetrating sar using cnns and gans," *Remote Sensing*, vol. 13, Feb. 2021.

[16] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[17] N. Campbell and J. Kautz, "Learning a manifold of fonts," *ACM Transactions on Computer Systems*, vol. 33, pp. 1–11, July 2014.

[18] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE Journal*, vol. 37, no. 2, pp. 233–243, 1991.

[19] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[20] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 1278–1286, PMLR, 22–24 Jun 2014.

[21] V. Kovenko and I. Bogach, "A comprehensive study of autoencoders' applications related to images," in *IT&I Workshops*, 2020.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] K. Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 06 2014.

[24] K. Yao, T. Cohn, K. Vylomova, K. Duh, and C. Dyer, "Depth-gated recurrent neural networks," 08 2015.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *ArXiv*, vol. abs/1810.04805, 2019.

[27] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.

[28] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *arXiv preprint arxiv:2006.11239*, 2020.

[29] P. Dhariwal and A. Q. Nichol, "Diffusion models beat GANs on image synthesis," in *Advances in Neural Information Processing Systems* (A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), 2021.

[30] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10674–10685, 2021.

[31] H. He, X. Chen, C. Wang, J. Liu, B. Du, D. Tao, and Y. Qiao, "Diff-font: Diffusion model for robust one-shot font generation," 12 2022.

[32] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds.), (Cham), pp. 234–241, Springer International Publishing, 2015.

[33] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *NIPS*, 2017.

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Los Alamitos, CA, USA), pp. 1–9, IEEE Computer Society, jun 2015.

[35] Y. Xie, X. Chen, L. Sun, and Y. Lu, "Dg-font: Deformable generative networks for unsupervised font generation," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Los Alamitos, CA, USA), pp. 5126–5136, IEEE Computer Society, jun 2021.

[36] R. G. Lopes, D. Ha, D. Eck, and J. Shlens, "A learned representation for scalable vector graphics," in *Deep Generative Models for Highly Structured Data, ICLR 2019 Workshop, New Orleans, Louisiana, United States, May 6, 2019*, OpenReview.net, 2019.

[37] A. Carlier, M. Danelljan, A. Alahi, and R. Timofte, "Deepsvg: A hierarchical generative network for vector graphics animation," 2020.

[38] V. Egiazarian, O. Voynov, A. Artemov, D. Volkhonskiy, A. Safin, M. Taktasheva, D. Zorin, and E. Burnaev, "Deep vectorization of technical drawings," in *Computer Vision – ECCV 2020*, (Cham), pp. 582–598, Springer International Publishing, 2020.

[39] T.-M. Li, M. Lukáč, G. Micha¨el, and J. Ragan-Kelley, "Differentiable vector graphics rasterization for editing and learning," *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, vol. 39, no. 6, pp. 193:1–193:15, 2020.

[40] P. Reddy, M. Gharbi, M. Lukac, and N. J. Mitra, "Im2vec: Synthesizing vector graphics without vector supervision," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Los Alamitos, CA, USA), pp. 7338–7347, IEEE Computer Society, jun 2021.

[41] G. Xie, X. Sun, X. Tong, and D. Nowrouzezahrai, "Hierarchical diffusion curves for accurate automatic image vectorization," *ACM Trans. Graph.*, vol. 33, nov 2014.

[42] X. Ma, Y. Zhou, X. Xu, B. Sun, V. Filev, N. Orlov, Y. Fu, and H. Shi, "Towards layer-wise image vectorization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2022.

[43] H. Aoki and K. Aizawa, "Svg vector font generation for chinese characters with transformer," in *2022 IEEE International Conference on Image Processing (ICIP)*, pp. 646–650, 2022.

[44] Y. Wang and Z. Lian, "Deepvecfont: Synthesizing high-quality vector fonts via dual-modality learning," *ACM Transactions on Graphics*, vol. 40, no. 6, 2021.

[45] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Commun. ACM*, vol. 27, p. 236–239, mar 1984.

[46] H. Hayashi, K. Abe, and S. Uchida, "Glyphgan: Style-consistent font generation based on generative adversarial networks," *Knowledge-Based Systems*, vol. 186, p. 104927, 08 2019.

[47] A. KumarBhunia, A. KumarBhunia, P. Banerjee, A. Konwer, A. Bhowmick, P. P. Roy, and U. Pal, "Word level font-to-font image translation using convolutional recurrent generative adversarial networks," *International Conference on Pattern Recognition*, Aug 2018.

[48] P. Lyu, X. Bai, C. Yao, Z. Zhu, T. Huang, and W. Liu, "Auto-encoder guided gan for chinese calligraphy synthesis," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, (Los Alamitos, CA, USA), pp. 1095–1100, IEEE Computer Society, nov 2017.

[49] S. Baluja, "Learning typographic style," *ArXiv*, vol. abs/1603.04000, 2016.

[50] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[51] Y. Jiang, S. Chang, and Z. Wang, "Transgan: Two pure transformers can make one strong gan, and that can scale up," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[52] J. Parente, L. Gonçalo, T. Martins, J. M. Cunha, J. Bicker, and P. Machado, "Using autoencoders to generate skeleton-based typography," in *Artificial Intelligence in Music, Sound, Art and Design* (C. Johnson, N. Rodríguez-Fernández, and S. M. Rebelo, eds.), (Cham), pp. 228–243, Springer Nature Switzerland, 2023.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[54] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[55] A. Borji, "Pros and cons of gan evaluation measures," *Computer Vision and Image Understanding*, vol. 179, pp. 41–65, 2019.

[56] L. S. WIRED, "Two legends dish on how to design a typeface," 2013.

[57] W. W. Royce, "Managing the development of large software systems: Concepts and techniques," in *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, (Washington, DC, USA), p. 328–338, IEEE Computer Society Press, 1987.

Appendices

Appendix A

ECAI 2023 Paper Submission

The following paper was written within the scope of this dissertation and was submitted to the 6th European Conference on Artificial Intelligence (ECAI 2023).

Improving Glyph Network Performance through Attention Loss Function and Input Selection: A Quantitative Analysis

First Author[a;*], Second Author[b] and Third Author[b]

[a]Short Affiliation of First Author
[b]Short Affiliation of Second Author and Third Author
ORCiD ID: First Author https://orcid.org/....-....-....-...., Second Author https://orcid.org/....-....-....-....,
Third Author https://orcid.org/....-....-....-....

Abstract.

Typography is one of the most impacting tools of graphic design since it is essential to communicate information effectively and create visually distinctive compositions. However, designing typefaces is often a time-consuming process that requires extensive experience and is therefore not always possible on projects with low budgets or of short duration. Recent developments in computer vision have made possible the development of robust generative models, such as Glyph Network, which is capable of inferring font style solely based on a small set of glyphs. This paper presents an approach to improve the state-of-the-art machine learning models to generate fonts from a small set of given glyphs. To do that, we improved the loss function of the original *GlyphNet* and determined the most informative input glyphs by applying weight masks to guide the training process according to characteristic details found on input glyphs. Furthermore, results reveal that an informed selection of the initial input subset leads to higher quality in outputs, and type designers' intuition regarding this choice is well supported by our ML-driven experiments. These findings compound to promising results above the current state-of-the-art and may be further extended with larger datasets as seen in other generative tasks.

1 Introduction

Historically, the manual design of new typographical fonts has proven to require a great deal of effort, even for skilled professionals [15]. Depending on the purpose and concept of the project, type designers need to define adequate styling features, such as width, height, weight, kerning, leading, and whether or not to use serifs or other adornments, among others. Moreover and even more complex is the process of designing congruent glyphs for all letters, numbers and other symbols that compose the target writing system [5]. For instance, designing and refining fonts often can take weeks or months after years of expertise, and even for creating geometric fonts, minute optical adjustments are often needed. Therefore, due to the lack of time and resources, creating tailor-made fonts is often not feasible in lower-budget design projects. And even with the availability of a large number of font families to end-users, editing minute details or ornaments might still be a necessity in many design projects, e.g. in the creation of new brands, as these must be unique and distinctive.

While early research tackled the problem of glyph generation through pipelined approaches that would analyze skeleton features and assemble new glyphs based on matched examples [19, 4, 10], recent developments in computer vision made possible the development of robust generative models, from GANs and Variational Autoencoders to Diffusion Models and Visual Transformers, that are capable of producing new, realistic-looking fonts [6, 14, 2, 24, 7, 12].

One such recent model is the Glyph Network developed by Azadi et al. [2], based on the conditional GAN architecture [11], which produces new glyphs in the same style as those entered by the users. Alongside the proposal for a new dataset of medium size for benchmarking typography generation systems, the original paper outlines the capability of *GlyphNet* to infer whole typefaces based on a few samples provided as input. However, It falls short of presenting objective, quantitative results for the performance of such a model, based on well-established metrics in the imaging domain.

Our Contributions. In this paper, we perform an extensive analysis of the Glyph Network output quality using multiple quantitative measurements and show how its performance can be improved by modifying the original loss function and selecting the most informative glyphs to use as input examples. The outline of the paper goes as follows. We start by summarizing the historical background of ML-based font generation systems in Section 2 and doing an overview of the network behind the originally proposed Glyph Network in Section 3. Conducted experiments and key conclusions are covered in sections 4 and 5, respectively. Code available at anonymous.4open.science/r/inproving-glyphnet-E78D.

2 Related Work

For the past few decades, many approaches have been proposed to tackle the font generation problem, where initial developments relied on the expertise of researchers to design glyph assembly processes based on skeleton and proportion features [20, 19, 4, 10]. While such approaches may suit more conventional fonts, i.e. designed according to more strict rules/topology, these cannot contemplate more unconventional typography, using unusual decorative elements or more

* Corresponding Author. Email: somename@university.edu.

experimental topology. Besides, such approaches have the shortcoming of being constrained and dependent on initial annotated data.

Meanwhile, the recent sharp rise in the popularity of deep neural architectures has brought paradigm shifts to many fields of Computer Science alongside significant improvements. CNNs and GANs, in particular, have proven to be powerful tools in pattern recognition problems and imaging domains. For example, these architectures can receive as input large quantities of glyph images and infer the styles and parameters that distinguish different fonts. As a consequence, they are capable of generating visual results that are more believable than the ones produced by shallow solutions [6, 2, 24, 7, 12]. Since typefaces are 2D objects that may be categorized based on factors like boldness, serifs or slope, most of the current solutions rely on encoder-decoder architectures to extract key latent features out of a single glyph to sample the remaining glyphs of the given font-family [14, 18, 16, 17].

However, the raster images produced by these models often come with perceptible degrees of noise and contour imperfections. Even in state-of-the-art work, this is a visible concern requiring manual tweaking and conversion to vector formats that properly scale to any necessary size before shipping the generated artifacts as standalone, final products.

Besides, there has been some work towards creating models for generating typefaces in scalable vector formats such as SVG. These are typically supported by either LSTMS [14, 21] or Transformers [1] and rely on annotated datasets of primitive shape and path sequences (e.g. *moveTo*, *lineTo*, *cubicBezier*, among others). These solutions are able to provide clean, scalable visuals for simple typefaces; however, their ability to replicate more complex and divergent typefaces remains unproven.

More recent developments include the appearance of Visual Transformers [12] and Diffusion Models [7] that outpace prior approaches with respect to visual output quality. The downsides of these models reside in the computational power demanded during training when tailored models to specific tasks are desired (i.e. font generation) and the large-scale datasets required to achieve results comparable to former deep convolutional models.

3 Methods

We propose key changes to the loss function and inputs of the state-of-the-art Glyph Network [2] in order to obtain trained models with better overall performance. We first summarize how the original *GlyphNet* operates and establish a baseline training setup in Section 3.1, followed by Sections 3.2 and 3.3 where new approaches to the training process are proposed and an overview of the dataset used across all experiments is made, respectively. Section 3.4 introduces the metrics employed to conduct the quantitative analysis and comparisons performed on the presented models.

3.1 Glyph Network

The Glyph Network is a Deep Convolutional GAN developed by Azadi et. al. [2] that is capable of generating all uppercase letters of a given font, for the Roman alphabet, based solely on a small subset of input glyphs. Its generator G1 is based on the image transformation network introduced by Johnson et. al. [13] and includes six ResNet blocks [8]. Figure 1 shows a broad illustration of this architecture.

Both inputs and outputs of the model consist of 26-channel font stacks, with each channel corresponding to a unique grayscale glyph in a 64x64 tensor. Some input glyphs are hidden, meaning that their corresponding channels are filled with null/zero values. The generator's goal is to reconstruct all 26 channels, including those that are not included as the input subset.

During backpropagation, 3 different loss components are taken into account: the LSGAN loss produced by the local discriminator D1, computed on individual glyphs of each stack; the LSGAN loss from global discriminator D2, computed on whole stacks at a time and the L1 loss, directly computed between the generated outputs of G1 and ground truth fonts. The full loss function is defined as follows:

$$\begin{aligned} \mathcal{L}(G_1) &= \lambda_1 \mathcal{L}_{L_1}(G_1) + \mathcal{L}_{LSGAN}(G_1, D) \\ &= \lambda_1 \mathcal{L}_{L_1}(G_1) \\ &+ \mathcal{L}^{local}_{LSGAN}(G_1, D_1) + \mathcal{L}^{global}_{LSGAN}(G_1, D_2), \end{aligned} \tag{1}$$

where G1 denotes the generator network, D1 and D2 refer to the local and global discriminators respectively and λ_1 is the weight of the L1 loss component.

During all the experiments outlined in Section 4, the hyperparameters of the Glyph Network and training optimizers were set to the values used by the original authors, as shown in Table 1. Due to computational constraints and in order to accommodate multiple initialization seeds, the number of training epochs was cut to half, from 600 to 300, except for the generation of the visuals of Figure 7.

Table 1. Hyperparameter values used during training in all experiments with Glyph Network.

Parameter	Value
Epochs	600
Batch Size	64
Training Pool	9121 fonts
Validation Pool	1472 fonts
Test Pool	1559 fonts
Learning Rate	0.0002
L_1 weight λ_1	100
Optimizer	Adam
LR Decay	Linear in last 100 epochs
Adam Momentum Decay β_1	0.5
Leaky ReLu Negative Slope	0.2

3.2 Attention based Loss

During early efforts to replicate the original model configuration, it was noted that it struggled to reproduce serifs and strokes of unusual typefaces, resulting in jagged, blurry glyph edges. These shortcomings are evidenced by the blue annotations in Figure 7. To guide the training process to retain the particularities of each target typeface, the L1 loss component was modified in order to accommodate attention maps of the same size as the model inputs (26x64x64). For each training example, a unique weight mask was built in order to attribute more importance to regions of the input tensor that contained distinctive font features. The final L1 loss component of the generator loss function is obtained by applying the dot-product of the attention maps with both network outputs and respective targets:

$$\mathcal{L}(G_1) = \lambda_1 \mathcal{L}_{L_1}(y_1 \cdot M, G_1(x_1) \cdot M), \tag{2}$$

where $G_1(x_1)$ is the network output given sample x_1 with target y_1 and M is the attention map. These three parameters are matrices of size (26x64x64).

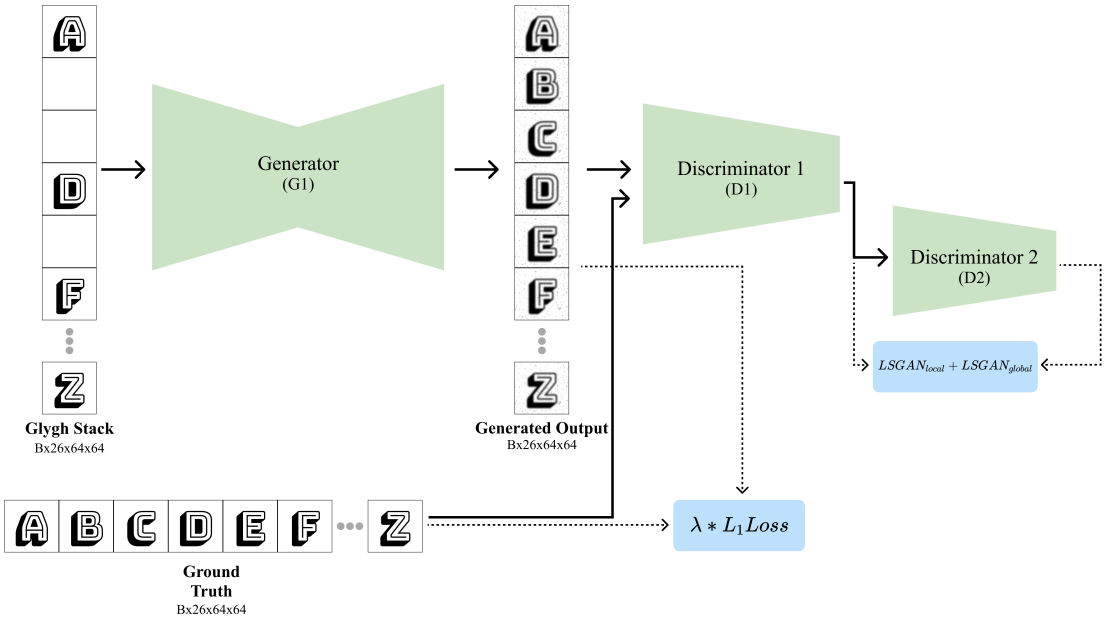

Figure 1. Glyph Network overview, retrieved from Azadi et. al. [2].

We explored the first version of these masks by computing the pixel-wise difference between a font devoid of ornaments (Code New Roman) and the target font. In a second version, these were obtained by applying the Zhang-Suen thinning algorithm [25] to each glyph, in order to bridge the flaws found in some glyph masks where they were very similar to the default font (i.e. resulting in an unusable attention mask). The Zhang-Suen thinning algorithm is an algorithm used to thin 2d objects found in binary images. When applied to glyphs, this thinning process leads to a close resemblance of glyph skeletons which can be further processed to identify joints and terminals. A few examples of these two processes are shown in Figure 2. In the Section 4, the three loss configurations will be referred to as $L1_{original}$, $L1_{attention}$ and $L1_{skeleton}$, respectively.

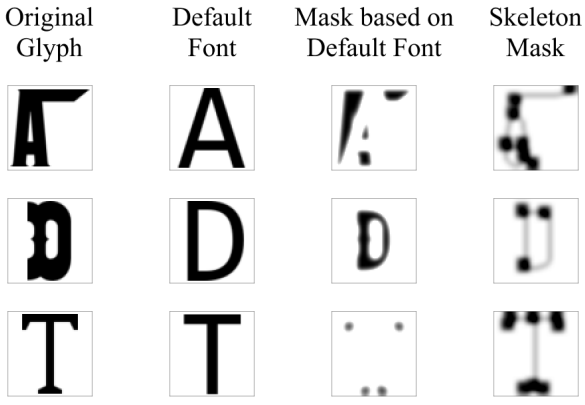

Figure 2. Examples of masks used for the computation of the L1 loss term. Original glyphs (1st column), default font with minimal ornaments (2nd column), the mask computed using the pixel-wise difference between glyphs of original and default font (3rd column) and mask obtained with the Zhang-Suen thinning algorithm (4th column)

3.3 *Dataset*

In order to achieve comparable quantitative results, the dataset used in all experiments was the *Capitals64* dataset collected by Azadi et al. [2]. This dataset includes 12K grayscale raster fonts with varying serifs, boldness, and textures, each composed of 26 ASCII uppercase letters framed within 64x64 canvases. The dataset is divided into train, test, and validation splits in a ratio of 10:1:1. A small data subset is shown in Figure 3.

Given that all symbols were fit to the same dimensions, some glyphs (i.e. W, M, T) found in the original fonts were deformed. Therefore, any model trained with this dataset is not equipped to learn features such as relative heights and sidebearings.

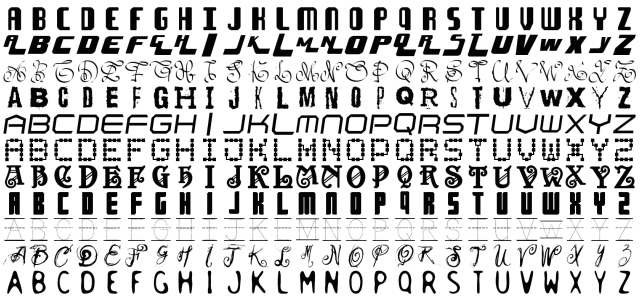

Figure 3. Train samples from *Capitals64* dataset.

3.4 *Evaluation Metrics*

In order to compare the performance of different models across the conducted experiments, FID [9], MSE and SSIM [22] were chosen as quantitative evaluation metrics. While SSIM is a metric used to measure the similarity between two images, according to luminance,

contrast and structure, MSE computes the mean square errors of pixel values and FID measures the difference between generated and real images in a distribution-wise manner with the aid of *InceptionNet*. MSE and SSIM are computationally less complex, while FID offers quantitative evaluation closer to the visual perception of humans [9, 3]. These are some of the metrics commonly employed in computer vision tasks and, more specifically, the ones used in the evaluation of current state-of-the-art approaches to font generation [18, 24, 2, 7]. Due to the stochastic nature of the training process, each model configuration was trained in up to 15 distinct RNG seeds and the mean values of each metric were computed on the test data.

4 Experiments and Results

In this section, a set composed of three experiments and subsequent analysis is presented, followed by an inspection of some representative visual artifacts and an overall review of the achieved results. Each experiment is accompanied by a complete quantitative analysis starting with Section 4.1 where three distinct approaches to the loss function are evaluated. In Section 4.2, we evaluate two distinct subsets, one with the best ranking glyphs as input to the model and the other is a subset typically defined by designers in the early stage of font creation. Finally, in Section 4.3, we study the impact of the size of the input subset based on the best-ranked subset from the previous experiment. All conducted experiments with stochastic nature were repeated 15 times with different random seeds. The following experiments were conducted on a single NVIDIA RTX 3080 Ti graphics card.

4.1 Letter Ranking

The three distinct L1 losses define three setups, $L1_{original}$, $L1_{attention}$ and $L1_{skeleton}$, which were analyzed by training the network with each one separately, feeding 3 randomly selected glyphs as input for each training sample. Subsequently, each resulting model was tested 26 separate times, in which only 1 glyph at a time was visible for the entire test pool. This experiment enabled the empirical search for the most informative letters in font generation contexts while simultaneously allowing the performance comparison between the three L1 loss configurations.

Results depicted in Figure 4 reveal that, when fed to the network individually, K, R and B contribute the most to the reconstruction of the original dataset while I, J and T are the least informative. This trend remained equal in all three loss setups and across both FID and MSE metrics. Moreover, when the training loss is directly tied to glyph-specific ornaments, *GlyphNet* achieves slightly better performance and the variance observed across both input letters and training seeds is decreased.

However, comparative results shown in Figure 4 (d) between $L1_{attention}$ and $L1_{skeleton}$ models are not conclusive since performance differences are inconsistent across different input glyphs. For example, given that the capital letter O does not contain joints or terminals in most typefaces, the attention mask computed using the Zhang-Suen Algorithm is not as effective, hence the decrease in performance. For that reason, the experiments that follow were conducted using the $L1_{attention}$ loss function.

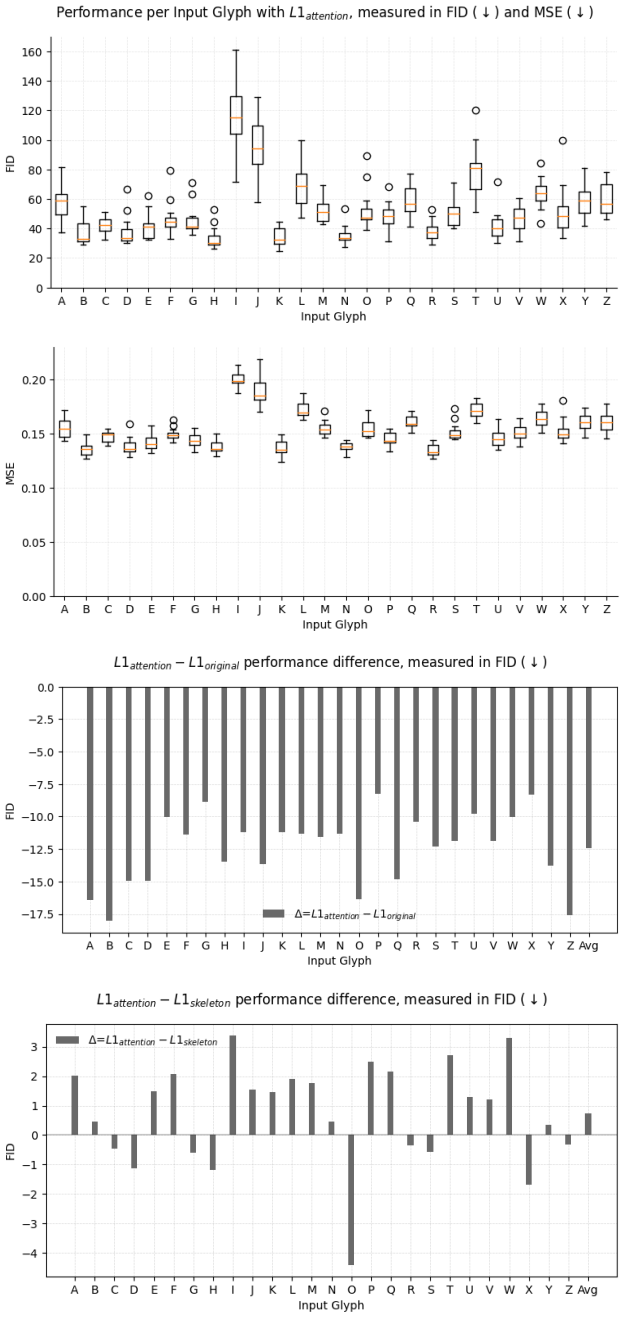

Figure 4. Letter Ranking with the $L1_{attention}$ model setup according to FID (↓), MSE (↓) in (a) and (b) respectively. Performance difference between different loss setups in (c) and (d).

4.2 ML vs Designers on Input Subset Choice

After studying how each individual glyph contributed to font inference, a subset of the 6 best-performing glyphs (B,D,K,H,N,R) was stacked to compare against a subset that a typeface designer typically starts the designing process with for a novel typeface (D,G,H,O,S,R) [23].

In this experiment, 15 different runs for the two groups of 6 letters were collected, each composed of models trained and tested solely on one of the input subsets. $L1_{attention}$ loss was used while all other setup elements remained unaltered.

Despite subset 1 being composed of the most informative letters found from the first experiment, it showed less precision across different training seeds and lower generative performance when com-

pared to subset 2, as shown in Table 2. The differences in performance measured with MSE between these two subsets are statistically significant according to the Wilcoxon Signed-rank test with a 95% confidence level ($p - value = 0.03125$).

Table 2. Glyph Network performance metrics per subset for the 15 different random seed runs with $L1_{attention}$.

	$\{K, H, B, N, R, D\}$			$\{H, O, D, S, G, R\}$		
	MSE(↓)	FID(↓)	SSIM(↑)	MSE(↓)	FID(↓)	SSIM(↑)
Mean	0.0611	7.4264	0.7283	0.059	6.5348	0.7372
StD	0.0027	1.5386	0.0117	0.0002	0.0701	0.0007

4.3 Input Subset Size

Since the original Glyph Network model can accommodate up to 26 input glyphs to generate a given font, it is rather relevant to observe how increasing the number of input glyphs impacts the overall performance of the model.

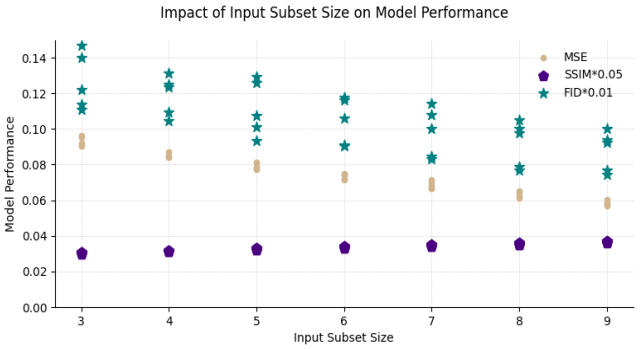

Figure 5. Performance evolution over increasingly larger input sizes. For each size, 15 distinct random seeds were used for network initialization.

For that reason, the network was trained and tested on increasingly bigger input sizes. Starting with the 3 best letters found in the Letter Ranking experiment and up to the top 9, the network was trained using $L1_{attention}$ loss up to 300 epochs. Its performance was measured with MSE, FID and SSIM metrics. Results computed across all three metrics, depicted in Figure 5, indicate the model performance increases in a linear fashion the more initial examples are given to it.

Although one might be tempted to increase the input size even further to obtain fonts with more quality, this strategy clashes with the initial goal of the presented system to aid typeface designers in the creation process of new fonts. If the user has already built a sizable collection of examples on its own, using a generative model to create the remaining characters, which in most cases need to be cleaned and revised, might be more time-consuming than assembling them by hand.

Finally, Figure 7 presents some output examples from a model trained over 600 epochs with $L1_{attention}$ loss and with the best input subset from Section 4.2 (D,G,H,O,S,R). As shown in Figure 7 by the best results produced with *GlyphNet*, the newly proposed $L1_{attention}$ approach also achieves better visuals in regard to ornament reconstruction. This is especially evident when comparing the

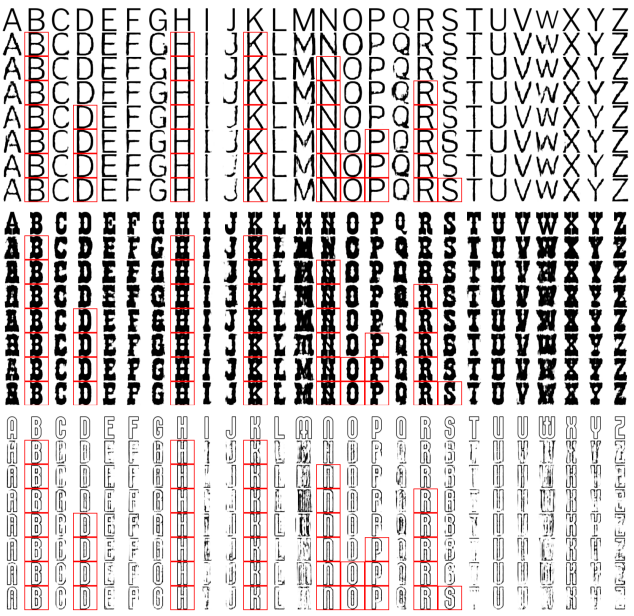

Figure 6. Visual examples drawn from the input subset size experiment where increasingly bigger input subsets were given to the Glyph Network. The target of each font is found in the top row while outputs of models with varying input sizes are presented in subsequent rows. Inputs are annotated in red.

glyphs generated with the original loss function, marked in blue, in which even simple strokes appear faded (see examples 1, 7 and 11 of Figure 7). These improvements, however, are not observed in the entirety of the test set (see example 9).

Overall, the Glyph Network excels at recreating fonts with thin, straight strokes, devoid of serifs and other ornaments. Furthermore, it is capable of identifying various degrees of boldness and italicization present in the inputs while capturing the general texture and style decisions of more complex typefaces to a certain degree.

5 Conclusions and Future Work

Typography is one of the most challenging tasks for designers, and in recent years, many automatic systems have been developed to aid the design of complete fonts based on a desired style. Amongst these approaches is included the Glyph Network model, which is one of the best machine learning algorithms in producing the most convincing results yet with as few input glyphs as possible. However, there are limitations when ornamented fonts are to be reconstructed; fonts with the original Glyph Network model approach tend to be incomplete on the ornaments and details. Moreover, there is an interest in the challenge of selecting the minimum input subset size of glyphs as input in order to generate the whole font while capturing the style and ornamentation of the target font.

In this paper, we improved the current state-of-the-art in font generation by modifying the existing Glyph Network loss function and carefully selecting its inputs during inference according to our thorough, objective analysis supported with multiple evaluation metrics. We systematically conducted experiments to evaluate the impact of each input Glyph and subsets composed of the best-evaluated ones and designers-picked ones. Furthermore, we also studied the impact of each glyph in the model and how the input subset size of glyphs impacts the performance metrics. In the end, the main contributions

Figure 7. Examples of *GlyphNet* outputs given the subset (D,G,H,O,S,R) as input for test samples. Each row is composed of the target font at the top with inputs annotated in red, followed by the generated outputs. Some undesirable artifacts found on inferred glyphs are marked in blue squares.

are a loss function that provides better results than the state-of-the-art approach, a selection of the best glyphs to produce cohesive fonts and the systematic analysis of the experiments that led to these contributions. The code for the approaches and experiments to generate the results of this paper is provided and available.

First, we established a baseline model and engineered two distinct attention mechanisms to compute the generator loss function: one based on pixel-wise operations, the $L1_{attention}$, and another supported by glyph skeletons obtained using a thinning algorithm, the $L1_{skeleton}$. We then conducted three experiments: a one-letter input ranking evaluation, a comparison of best ranking glyphs subset from the first experiment with a preset of select glyphs that designers start with when designing a novel typeface and; a systematic comparison of the impact of the input subset size in the model. In the letter ranking experiment, we showed that the $L1_{attention}$ loss led to a significant increase in all glyphs when compared to the $L1_{baseline}$, the state-of-the-art approach, in the FID metric, on average a total of 12.5 points decrease in the FID metric (lower is better). Although $L1_{skeleton}$ might bring slightly more stable and better results for most glyphs, it struggled in the ones where skeleton features were not as obvious to compute. The experiments conducted with multiple input glyphs served to optimize even further the performance of the Glyph Network while still maintaining a relatively low number of initial subset glyphs required to generate each font. These experiments also offered interesting insights into the potential performance benefits of incorporating human design intuition into machine learning systems. By studying how each individual glyph impacts the quality of the synthesized fonts, this analysis is expected to offer key valuable insights for future research on font generation systems. We conclude that the proposed Attention Loss for the Glyph Network is a viable and suitable solution for generating new and realistic-looking glyphs. Some of the fonts would require some pre-processing, but with the attained results, we are accelerating the design process of typeface artists by providing a general view of their creations in the very early stages of the process.

For future work, we plan on expanding the attention model to learn to detect the ornaments, i.e. learn the masks used for the $L1_{attention}$ loss. We aim to expand to datasets with more ornamented Glyphs, in this study we compared our approach with the state-of-the-art approach and used the same datasets for fair comparison. Furthermore, we plan on testing all combinations of available subsets of glyphs that yielded better results as well the subset used by designers.

References

[1] Haruka Aoki and Kiyoharu Aizawa, 'Svg vector font generation for chinese characters with transformer', (06 2022).

[2] Samaneh Azadi, Matthew Fisher, Vladimir Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell, 'Multi-content gan for few-shot font style transfer', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 11, p. 13, (2018).

[3] Ali Borji, 'Pros and cons of gan evaluation measures', *Computer Vision and Image Understanding*, **179**, 41–65, (2019).

[4] NDF Campbell and J Kautz, 'Learning a manifold of fonts', *ACM TRANSACTIONS ON GRAPHICS*, **33**, (07 2014).

[5] K. Cheng, *Designing Type*, Laurence King, 2006.

[6] Hideaki Hayashi, Kohtaro Abe, and Seiichi Uchida, 'Glyphgan: Style-consistent font generation based on generative adversarial networks', *Knowledge-Based Systems*, **186**, 104927, (2019).

[7] Haibin He, Xinyuan Chen, Chaoyue Wang, Juhua Liu, Bo Du, Dacheng Tao, and Yu Qiao. Diff-font: Diffusion model for robust one-shot font generation, 12 2022.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, 'Deep residual learning for image recognition', pp. 770–778, (06 2016).

[9] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter, 'Gans trained by a two time-scale update rule converge to a local nash equilibrium', in *Advances in Neural Information Processing Systems*, eds., I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, volume 30. Curran Associates, Inc., (2017).

[10] Phan Huy, Hongbo Fu, and Antoni Chan, 'Flexyfont: Learning transferring rules for flexible typeface synthesis', *Computer Graphics Forum*, **34**, (10 2015).

[11] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei Efros, 'Image-to-image translation with conditional adversarial networks', pp. 5967–5976, (07 2017).

[12] Yifan Jiang, Shiyu Chang, and Zhangyang Wang. Transgan: Two transformers can make one strong gan, 02 2021.

[13] Justin Johnson, Alexandre Alahi, and Li Fei-Fei, 'Perceptual losses for real-time style transfer and super-resolution', in *Computer Vision – ECCV 2016*, eds., Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, pp. 694–711, Cham, (2016). Springer International Publishing.

[14] Rapha Gontijo Lopes, David Ha, Douglas Eck, and Jon Shlens, 'A learned representation of scalable vector graphics', (2019).

[15] E. Lupton, *Thinking with Type*, Princeton Architectural Press, 2010.

[16] Pengyuan Lyu, Xiang Bai, Cong Yao, Zhen Zhu, Tengteng Huang, and Wenyu Liu, 'Auto-encoder guided gan for chinese calligraphy synthesis', *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, **01**, 1095–1100, (2017).

[17] Jéssica Parente, Luís Gonçalo, Tiago Martins, João Miguel Cunha, João Bicker, and Penousal Machado, 'Using autoencoders to generate skeleton-based typography', in *(to be published in) Artificial Intelligence in Music, Sound, Art and Design – 12th International Conference, EvoMUSART 2023, Held as Part of EvoStar 2023, Brno, Czech Republic, April 12-14, 2023, Proceedings*, eds., Colin Johnson, Nereida Rodríguez-Fernández, and Sérgio M. Rebelo. Springer, (2023).

[18] Donghui Sun, Qing Zhang, and Jun Yang, 'Pyramid embedded generative adversarial network for automated font generation', pp. 976–981, (08 2018).

[19] Rapee Suveeranont and Takeo Igarashi, 'Example-based automatic font generation', in *Smart Graphics*, eds., Robyn Taylor, Pierre Boulanger, Antonio Krüger, and Patrick Olivier, pp. 127–138, Berlin, Heidelberg, (2010). Springer Berlin Heidelberg.

[20] Rapee Suveeranont and Takeo Igarashi, 'Example-based automatic font generation', in *Smart Graphics*, eds., Robyn Taylor, Pierre Boulanger, Antonio Krüger, and Patrick Olivier, pp. 127–138, Berlin, Heidelberg, (2010). Springer Berlin Heidelberg.

[21] Yizhi Wang and Zhouhui Lian, 'Deepvecfont: Synthesizing high-quality vector fonts via dual-modality learning', *ACM Trans. Graph.*, **40**(6), (dec 2021).

[22] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli, 'Image quality assessment: from error visibility to structural similarity', *IEEE Transactions on Image Processing*, **13**(4), 600–612, (2004).

[23] Liz Stinson WIRED. Two legends dish on how to design a typeface, 2013.

[24] Yangchen Xie, Xinyuan Chen, Li sun, and Yue lu, 'Dg-font: Deformable generative networks for unsupervised font generation', in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (2021).

[25] T. Y. Zhang and C. Y. Suen, 'A fast parallel algorithm for thinning digital patterns', *Commun. ACM*, **27**(3), 236–239, (mar 1984).

Appendix B

Artificial Fonts

Below is an extended sample of generated fonts using both the diffusion model and GlyphNet addressed in this dissertation. For each font, the first row contains the original font with annotated inputs in red, followed by the output produced by GlyphNet with $L1_{attention}$ loss, and the last row presents the glyphs generated using the diffusion model. The original fonts belong to the Capitals64 test pool.