NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Eivind Yu Nilsen

Few-shot Font Style Transfer with Extraction of Partial Style

Master's thesis in Computer Science
Supervisor: Björn Gambäck
June 2022

Eivind Yu Nilsen

Few-shot Font Style Transfer with Extraction of Partial Style

Master's thesis in Computer Science
Supervisor: Björn Gambäck
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Eivind Yu Nilsen

Few-shot Font Style Transfer with Extraction of Partial Style

Master's thesis in Computer Science, June 2022
Supervisor: Björn Gambäck

Data and Artificial Intelligence Group
Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

Abstract

Text is a prominent visual element for conveying ideas and emotions, and common occurrence in our daily lives. As the representation of texts, fonts play an essential role. Fonts show whether a text is serious or casual, scary or playful. They can impact the quality of a joke, or make sentences more memorable. However, designing fonts can be very time consuming, especially for languages with higher character count. Few-shot font style transfer is able to automate this process, using only a few samples as reference.

This report proposes a novel font style transfer method, named few-shot font style transfer with Extraction of Partial Style (EPS-Font). The method attempts to solve font style transfer differently than a typical way, which is using the same encoder architecture for content feature extraction and style feature extraction, with an encoder architecture that focuses on treating the style reference images as *partial* style reference images. It is shown that this way of extracting style features greatly increases the results over the typical way, and that EPS-Font is able to outperform state-of-the-art methods in both quantitative and qualitative evaluations. Additionally, experiments on using Deep Metric Learning (DML) for font style transfer are conducted. The results of these experiments suggest that DML does not improve the performance of the model, and that this is because the model is affected negatively by DML when the datasets have many similar fonts. Lastly, a modification that can be applied into a font style transfer method, named Deformation and Texture Separation (DTS) is presented. This modification separates the task of font style transfer into two parts: predicting the deformation and predicting the texture using the deformation. DTS shows interesting results and is a potential approach for new font style transfer methods.

Sammendrag

Tekst er en sentral verktøy for å formidle ideer og følelser, og er en svært vanlig forekomst i våre liv. Som representasjon av tekst spiller skrifttype en viktig rolle i formidlingen. Skrifttyper viser om en tekst er alvorlig eller uformell, skummel eller leken. De kan påvirke kvaliteten på en vits, eller gjøre setninger mer minneverdige. Design av skrifttyper kan imidlertid være svært tidkrevende, spesielt for språk med høyere tegnantall. En teknikk kalt skriftstiloverføring kan brukes til å automatisere denne prosessen ved å bruke bare noen få eksempler som referanse. Skriftstiloverføring utføres ved å ekstrahere tegnet fra et bilde og ekstrahere skrifttypen fra referansebilder, og blande disse ekstrasjonene sammen for å lage et bilde av samme tegn og skrifttype som de respektive kildene.

Denne rapporten foreslår en ny metode for skriftstiloverføring, kalt few-shot font style transfer with Extraction of Partial Style (EPS-Font). Metoden forsøker å løse problemet annerledes enn en typisk måte, som er å bruke samme kodearkitekturen for ekstrasjon av tegnet og for ekstrasjon av skrifttypen, med en koderarkitektur som istedet behandler referansebildene av skrifttypen som *delvis* referansebilder. Det er vist at denne måten å ekstraktere skrifttypen i stor grad øker resultatene fremfor den typiske måten, og at EPS-Font er i stand til å utkonkurrere state-of-the-art metoder i både kvantitative og kvalitative evalueringer. I tillegg utføres eksperimenter med å bruke Deep Metric Learning (DML) for skriftstiloverføring. Resultatene av disse eksperimentene tyder på at DML ikke forbedrer ytelsen til modellen, og at dette er fordi modellen påvirkes negativt av DML når datasettene har mange like skrifttyper. Til slutt presenteres en modifikasjon som kan brukes i problemet, kalt Deformation and Texture Separation (DTS). Denne modifikasjonen deler skriftstiloverføringsoppgaven i to deler: gjette utformingen til bokstaven, så gjette teksturen ved hjelp av utformingen. DTS viser interessante resultater og er en potensiell retning for nye metoder.

Preface

This report was written at the Department of Computer Science at the Norwegian University of Science and Technology (NTNU). The work was done under the supervision of Björn Gambäck for the course TDT4900 - Computer Science, Master's Thesis.

This thesis reuses parts of my report in TDT4501 - Computer Science, Specialization Project, and a report by Eivind Rebnord and I in TDT12 - Computational Creativity. This includes some sentences in the abstract. It will be mentioned at the beginning of a chapter whenever parts of these reports are being reused.

I would like to thank Björn Gambäck for his supervision and feedback throughout my project. I would also like to thank Tian Yuchen, Song Park and Florian Schroff for giving me permission to use their figures. Furthermore, I would like to thank Eivind Rebnord for the collaboration on the report in TDT12.

The code for this thesis is available at: `https://github.com/EivindYN/EPS-Font`

Eivind Yu Nilsen
Trondheim, 30th June 2022

Contents

List of Figures

List of Tables

1. Introduction

In this chapter, an insight into this thesis is given. The goal of this work is to progress the state-of-the-art performance of few-shot font style transfer, with a focus on making unrecognizable characters infrequent. The background and motivation of few-shot font style transfer are described in Section 1.1, which is partially based on the specialization project mentioned in the preface. The goal and research questions, which will be answered in this thesis, are presented in Section 1.2. The main contributions of this thesis is given in Section 1.3. Finally, an overview of this thesis is given in Section 1.4 with brief explanations of each chapter.

1.1. Background and Motivation

Designing fonts can be time-consuming, especially for languages with higher character count. For instance, the national encoding standard character set issued by the Chinese government in 2000 consists of 20,000+ Chinese characters[1]. Many researchers have, therefore, proposed approaches to automate this (Yuchen, 2017; Azadi et al., 2018; Park et al., 2021b; Li et al., 2021). In the early work on font generation, methods mainly relied on shape modeling of outlines and interpolation (Suveeranont and Igarashi, 2010; Campbell and Kautz, 2014). While these methods got good results, they were easily overwhelmed when performing on font styles with more low-level handcrafted features (e.g., graphical or geometrical features). Recently, deep neural networks (DNNs) have spurred a lot of interest in this research field. Methods using DNNs focus on being able to generate font libraries by observing a subset of them. Specifically, these methods use pre-trained models, usually trained on a large dataset, then the pre-trained models are fine-tuned on a subset of the fonts that are to be generated. While these deep learning approaches have achieved higher performance than previous methods, there are still some challenges remaining. Firstly, having to do a fine-tuning for every font library that is to be generated limits the real-time practicability. Secondly, fine-tuning needs several hundred samples, in some cases thousands (Yuchen, 2017). This makes it ineffective for alphabets with low character counts such as the Latin alphabet. Additionally, making these samples is a labor-intensive job.

Recently, many few-shot learning methods have been proposed. These methods aim at generating font libraries using only a few samples, without any additional fine-tuning at test time. Typically, state-of-the-art few-shot font generation methods solve the task as a few-shot font style transfer task, disentangling the content and style features from glyph

[1] https://docs.oracle.com/cd/E19253-01/817-2523/auto25/index.html

images and combining the features to generate a prediction (Gao et al., 2019; Park et al., 2021b; Li et al., 2021; Xie et al., 2021). The state-of-the-art methods in few-shot font style transfer have shown very good results, but are not fully rid of defects. Even small inaccuracies decrease the incentive of using these methods. Errors like unrecognizable characters would arguably make an application look incompetent, similar to how spelling mistakes make a writer look unprofessional. Therefore the aim of this work is to push the state-of-the-art to a level where unrecognizable characters are infrequent, instead of somewhat recurrent, to further push the practical usability of this field. Additionally, a big motivation is opening up the possibility of using font style transfer for real-time translation with maintenance of font style. Therefore, the thesis focuses on font style transfer between different languages.

1.2. Goals and Research Questions

The goal of this project is:

Goal *Progress the state-of-the-art in few-shot font style transfer between different languages such that unrecognizable characters are infrequent, even in highly stylized fonts*

The goal is to make a font style transfer method that feels safe to use real-time, that is, without the risk of generating unrecognizable characters. One approach to achieve this goal is to push the state-of-the-art's overall performance in addition to the generated characters' recognizability. Another approach is to improve the generated characters' recognizability at the sacrifice of the overall performance. If so, an analysis of whether these sacrifices are worth it should be performed.

To reach this goal the following research questions (RQs) will be addressed:

RQ1 *How should the content feature and the style feature be properly extracted in few-shot font style transfer?*

Content feature and style feature extraction are crucial parts of any style transfer method. Identifying proper ways for extracting the respective feature types is beneficial for new designs of few-shot font style transfer methods.

RQ2 *How can Deep Metric Learning be beneficial to few-shot font style transfer?*

Deep Metric Learning (DML) has previously been applied to font style transfer by Aoki et al. (2021), however it has received very little acknowledgement as of writing this thesis. The effect of DML will be evaluated with a recently proposed method by Xuan et al. (2020), which shows promising results and is described in detail in Chapter 4.

RQ3 *How can the deformation and texture in few-shot font style transfer be separated?*

With deformation we refer to the difference of shape (e.g. cursive, big/small size, artistic traits). A separation of deformation and texture would open up more possibilities for training few-shot font style transfer models. The goal with this research question is to find a mapping that with the input of a glyph image x and a reference image c can either:

- Output the glyph image x with the same *deformation* as the reference image c.
- Output the glyph image x with the same *texture* as the reference image c.

1.3. Contributions

This thesis makes the following contributions:

1. Proposes a simple yet effective few-shot font style transfer model, few-shot font style transfer with Extraction of Partial Style (EPS-Font). This solves few-shot font style transfer by treating the style reference images as *partial* style reference images.
2. Showcases the effect of Deep Metric Learning for few-shot font style transfer.
3. Proposes a modification that can be applied to a few-shot font style transfer model to separate deformation and texture prediction.
4. Presents a detailed overview of the fundamentals and state-of-the-art for font generation.

1.4. Thesis Structure

The remainder of this thesis is structured as follows:

- Chapter 2 covers basic knowledge of neural networks and essential knowledge for understanding state-of-the-art few-shot font style transfer methods.
- Chapter 3 gives an overview of previous work in font style transfer, in addition to state-of-the-art methods that are related to the proposed method.
- Chapter 4 describes the proposed method, few-shot font style transfer with Extraction of Partial Style (EPS-Font) and the proposed modification for few-shot font style transfer, named Deformation and Texture Separation (DTS).
- Chapter 5 describes how the experiments will be evaluated and the setup for each experiment, and then presents the experimental results.
- Chapter 6 discusses the results of the experiments and merits of the work conducted as well as limitations.
- Chapter 7 discusses the main contributions made to the field and discusses where to extend or improve this work.

2. Background Theory

This chapter will cover essential knowledge for few-shot font style transfer. Section 2.1 describes the basics of neural networks. Section 2.2 describes CNN, a class of neural network that is most commonly applied to analyze visual imagery. Sections 2.3-2.6 describe generative adversarial network (GAN), a framework for training a neural network at being ”realistic” by tasking it with tricking another neural network, and other theory that relate to GAN. Section 2.7 describes style transfer, a technique that transfers the style of a set of images onto a content image. Section 2.8 describes image-to-image translation, a task of translating images from one domain to another. Section 2.9 describes Deep Metric Learning, a group of techniques that aim to establish similarity or dissimilarity between embeddings based on their semantic data.

Sections 2.1-2.2 and Section 2.7 are rewritten and Figure 2.3 is reused from the specialization project mentioned in the preface. Section 2.3 is rewritten from the report in TDT12 mentioned in the preface.

2.1. Artificial Neural Network

Artificial neural networks (ANN) is a computing system inspired by biological neural networks, taking inspiration from the way neurons communicate and are connected in a biological brain. The simplest form of ANN can be delineated in the form of a single formula (Equation 2.1)(Russell and Norvig, 2010). The term x is the set of input values $\{x_1, x_2...x_n\}$, w is the set of corresponding weights $\{w_1, w_2...w_n\}$, b is the bias and $\hat{y}$ is the output value.

$$\hat{y} = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

In order for an ANN to learn, it modifies the weights and bias. Weights decide how much each input value contributes to the output value. The bias is a value added or subtracted from the output value. The goal of an ANN is to modify the weights and bias to create a function that maps the input x to the target output y. An optimization technique commonly used to train ANNs is gradient descent (Russell and Norvig, 2010).

Gradient descent Gradient descent is an optimization algorithm which iteratively minimizes some function using the direction of steepest descend, the negative gradient (Russell and Norvig, 2010). To do gradient descent, a calculation of the loss is required. A common way to calculate loss is to use the squared loss function based on the predicted output $\hat{y}$ and the target output y defined as $L = (\hat{y} - y)^2$. Then calculate the negative gradient, the change of value which decreases this loss the most. For the squared loss function that is $\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$. Next modify weights and bias using the negative gradients. A good example to visualize gradient descent is a 2D hill (Figure 2.1). The green arrows showcase the direction of the slope (the negative gradient) each ball is laying on. If ball A or B follow the direction of their slope until it reaches a flat surface, it reaches a global minimum. If ball C follows the direction of its slope until it reaches a flat surface, it reaches a local minimum.

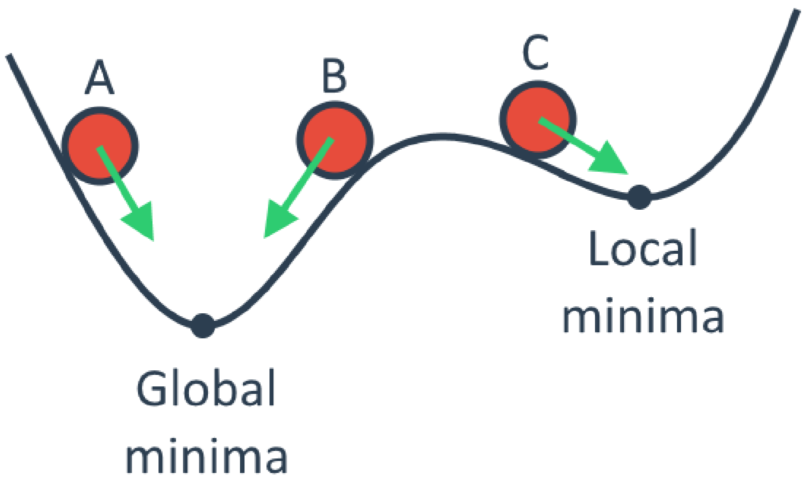


Figure 2.1.: Example of gradient descent in 2D

Activation function Sometimes it can be beneficial to control the range of the output value. To do so, one would use an activation function. For instance, if you make a neural network that were to predict the likelihood of some event, the outputted number should be limited between 0 and 1. For this scenario, one could use the sigmoid activation function, which maps $\mathbb{R} \rightarrow (0, 1)$ and is defined as $S(x) = \frac{1}{1+e^{-x}}$ (Russell and Norvig, 2010). The most widely-used activation function, since 2017, is Rectified Linear Unit (ReLU) (Ramachandran et al., 2018), which is defined as $f(x) = x^+ = max(0, x)$.

Multiple layers In order to allow for more complex function approximations, extra layers can be included between the input and output layer which are called hidden layers (Russell and Norvig, 2010). When an ANN has hidden layers, it is also called a deep neural network (DNN). An example of a complex function that a DNN can perform which an ANN is unable to is exclusive disjunction, a logical operation that is true if and only if its arguments are different. An example of a DNN is shown in Figure 2.2. To train a DNN, you calculate the gradients in a backward manner, starting from the output neuron. This is because the gradient from one layer can be reused in the computation of the gradient for the previous layer, and allows for efficient computation of gradients. This algorithm for training a DNN is called backpropagation (Russell and Norvig, 2010).

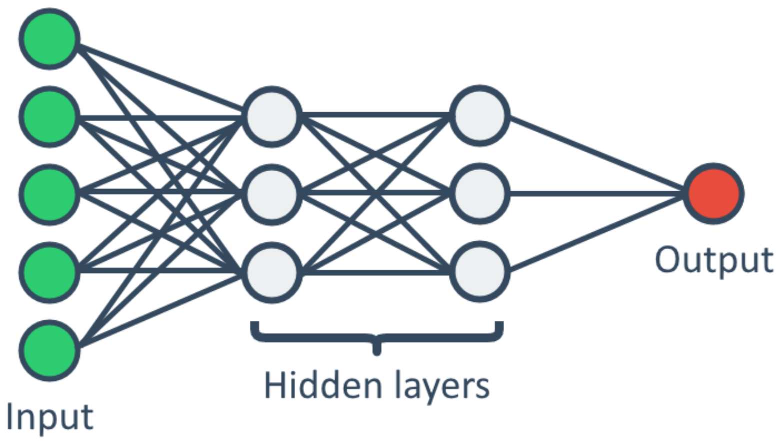


Figure 2.2.: Example of a neural network

Normalization Normalization is a common machine learning technique which standardizes a data distribution. This is commonly used between layers in a neural network, as it is useful for preventing the distribution from shifting between layers. Shifts in distribution are problematic for the training, as they could constantly need to be adjusted for every layer, which would essentially become a forever moving target. A typical normalization technique is batch normalization, introduced by Ioffe and Szegedy (2015). Batch normalization alleviates domain shifts, and addresses the issue of gradients that explode or vanish, as normalizing activations throughout the networks prevent small changes from amplifying to large and suboptimal changes. The batch normalization standardizes the data distribution for each feature channel (Huang and Belongie, 2017):

$$BN(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta, \tag{2.2}$$

where γ and β are affine parameters learned from data, $\{\mu(x), \sigma(x)\}$ are the mean and the standard deviation of the data distribution across batch size and spatial dimensions for each feature channel:

$$\mu_c(x) = \frac{1}{NHW} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw} \tag{2.3}$$

$$\sigma_c(x) = \sqrt{\frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw} - \mu_c(x))^2 + \epsilon}, \tag{2.4}$$

where ϵ is some small value to prevent division by zero in Equation 2.2.

2.2. Convolutional Neural Network

Convolutional neural network (CNN) is a class of neural networks, commonly applied to images (He et al., 2016; Isola et al., 2017; Huang and Belongie, 2017; Zhang et al., 2019; Brock et al., 2018). CNN reduces the number of parameters necessary to train by reusing its weights at different spatial locations and makes better use of spatial and temporal properties (Burkov, 2019).

CNN uses a kernel, which are weights organized in a $h \times w$ spatial dimension, where h is the height of the kernel, and w is the width. The kernel is propagated throughout the image in a overlapping fashion, starting from the top-left corner (Burkov, 2019). The output generated is often referred to as a feature map (Goodfellow et al., 2016). In the first layers of a CNN, the feature map will consist of simple shapes like edges and lines. The deeper layers will combine the earlier shapes to create more advanced shapes, for example a circle, faces and different textures. Since there is a bigger variety of advanced shapes compared to simple shapes, the deeper layers typically have a bigger size of features (called depth) per location in the feature map.

2.3. Generative Adversarial Network

Generative adversarial network (GAN) is a machine learning framework designed by Goodfellow et al. (2014). GAN consists of two neural networks, a generator G and a discriminator D, that compete against each other in a "game". G has the task of generating a plausible instance given the dataset, for instance an image, while D has the task of classifying whether an instance is from the dataset (not generated by G) (Goodfellow et al., 2014). An overview of the general GAN architecture is shown in Figure 2.3.

In order for G to diversify its outputs, it is given a noise vector z. A generated instance can therefore be represented as $G(z)$. $D(y)$ represents D's classification of an instance y. D's task is to maximize $log(D(y))$ when y is from the dataset, and minimize $log(D(y))$ when y is generated from G. G's task is to minimize $log(1 - D(G(z)))$, "tricking" the discriminator. The objective function becomes (Goodfellow et al., 2014):

$$\min_G \max_D V(D, G) = \mathbb{E}_y[\log D(y)] + \mathbb{E}_z[\log(1 - D(G(z)))] \tag{2.5}$$

A benefit of using GAN is that it makes it possible to train the model without any ground truth. For instance, a GAN made to generate faces could generate the face of a person that has never existed. However, even in situations where there are ground truths,

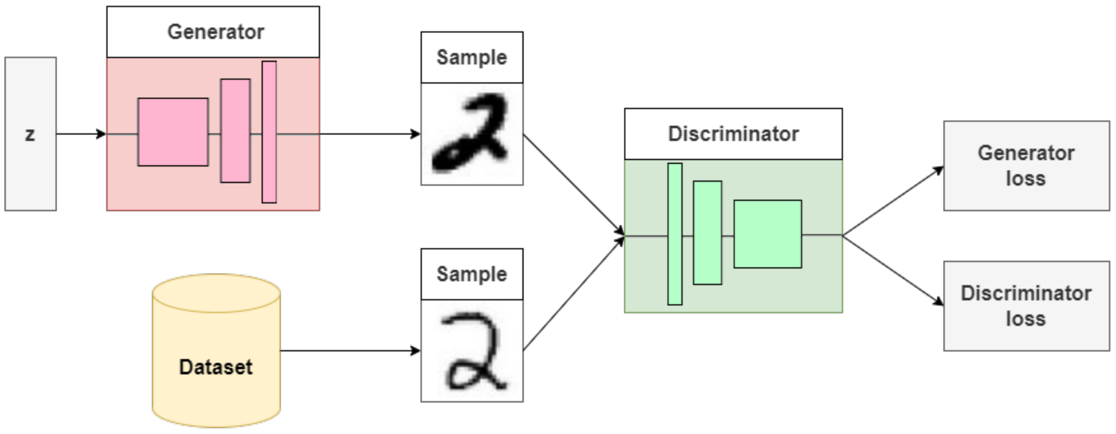


Figure 2.3.: Example of a standard GAN architecture

GAN is still useful as it encourages the model to generate realistic and sharp instances to fool the discriminator. An example of a good use case for GAN for training with ground truth images is training a model with both GAN and L1 loss. L1 loss is known to be good at low frequency details, but also cause blurry results(Pathak et al., 2016; Zhang et al., 2016), because it averages all plausible outputs. However, blurry images will be easily recognized by the discriminator. Therefore, the generator is trained by the discriminator to product realistic and sharp outputs. An example that can be hard to learn with L1 loss, but without GAN, is patterns that look seemingly random, like grass, freckles, or hair straws.

2.4. Conditional Generative Adversarial Network

GANs are able to generate a realistic face given the noise input z, but the user does not have any control over characteristics of the face generated (e.g. hair color or gender). In order to be able to control these characteristics in the output, one can use conditional GANs (cGAN). Instead of mapping $z \rightarrow y$, an additional condition x is applied to the input, like a class label or image for inpainting (Isola et al., 2017). This results in the mapping $\{x, z\} \rightarrow y$. With this change, the objective function becomes (Mirza and Osindero, 2014):

$$\min_G \max_D V(D, G) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(G(x, z)))] \tag{2.6}$$

2.5. Hinge Loss

Hinge loss is a loss function which was introduced by Gentile and Warmuth (1998). It has earlier shown to be of great use for classification (Rosasco et al., 2004), however it has not been notable in the field of GAN until recently. Presumably, the introductions of hinge loss in many recent state-of-the-art techniques featuring GAN (Miyato et al., 2018; Zhang et al., 2019; Brock et al., 2018) has now made it prominent in the field. The hinge version of the adversarial loss is defined as follows (Zhang et al., 2019):

$$\begin{aligned} L_D = &- \mathbb{E}_{x,y}[-1 + D(x,y)]_+ \\ &- \mathbb{E}_{x,z}[-1 - D(x, G(x,z))]_+ \\ L_G = &- \mathbb{E}_{x,z}[D(x, G(x,z))] \end{aligned} \tag{2.7}$$

2.6. Fréchet Inception Distance

To evaluate the results of generated images, one can use the Fréchet Inception Distance (FID) score (Heusel et al., 2017). The score tells the similarity between real data x and generated data $\hat{x}$, with lower scores meaning they are more similar. The way the FID score is calculated is by propagating the real data and the generated data through the Inception-v3 model (Szegedy et al., 2016) and then calculate the mean μ and covariance matrix Σ of the features of the embeddings, specifically one of the deepest embeddings. Embeddings are outputs from layers before the output layer in neural networks. Next the mean and covariance matrix are assumed to follow a multidimensional Gaussian, so Fréchet Distance is used to calculate the difference between these Gaussians by (Heusel et al., 2017):

$$FID(x, \hat{x}) = ||\mu_x - \mu_{\hat{x}}||_2^2 + \mathrm{Tr}\,(\Sigma_x + \Sigma_{\hat{x}} - 2(\Sigma_x \Sigma_{\hat{x}})^{1/2}) \tag{2.8}$$

where Tr is the sum of the elements in the diagonal. The authors of ”GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”, Heusel et al. (2017), which proposes the FID score, recommend using a minimum sample size of 10,000 for calculating the FID.[1]

2.7. Style Transfer

Style transfer aims to learn to transfer the style from one image to another. Gatys et al. (2016) performed style transfer by training a CNN at making an individual feature space representation for the style and the content of an image. Then by using the style representation s of an image, and content representation c of another image, you could synthesise an image with content c and style s, as a result performing style transfer.

Font style transfer Font style transfer is a technique for performing font generation. The task of font style transfer can be defined with the same mapping as for style transfer, $\{c, s\} \rightarrow x$, that is generating a target image x given the content of c and the style of s. However, for style transfer the style is often regarded as a set of colors and texture (Gatys et al., 2016; Johnson et al., 2016; Chen and Schmidt, 2016), while for font style

[1] https://github.com/bioinf-jku/TTUR

transfer the style is more abstract. For instance, a font style can differ from another by stroke, shape or decorations. Therefore, style transfer methods can not be directly applied to font style transfer.

2.8. Image-to-image Translation

Image-to-image translation is the task of translating images from one domain to another, given sufficient training data. For instance, translating a grayscale image to colored image. Isola et al. (2017) proposed pix2pix, a popular image-to-image model which introduced cGAN to the field. They trained the cGAN by using input-output pairs $\{x, y\}$, where the generator G was tasked at mapping $x \rightarrow y$, while the discriminator D was tasked at differentiating between y and the output of G. Isola et al. (2017) made a number of changes to the cGAN architecture. The generator was changed from an encoder-decoder structure to a U-Net structure (Ronneberger et al., 2015) and the discriminator architecture was changed to what they named "PatchGAN". The U-Net structure adds connections between layers of the encoder and the decoder, making them able to share low-mid level features. This is highly useful for image-to-image, as in many problems, the input and output share low level information. For instance, the location of prominent edges. The PatchGAN discriminator discriminates between real and fake local patches, as opposed to the whole image. This exposed the discriminator to only mid-high frequency details, making it encourage sharp and realistic images, even if incorrect.

2.9. Deep Metric Learning

Deep Metric Learning (DML) is a machine learning approach where the goal is to learn a representation function $f(x)$ that maps data x into a feature space $\mathbb{R}^D$, whereas data that are semantically similar are metrically close in $\mathbb{R}^D$, while data that are semantically different are metrically distant in $\mathbb{R}^D$ (Hermans et al., 2017; Xuan et al., 2020). Analogously, DML aims at optimizing the embedding itself, rather than an intermediate bottleneck layer. To achieve this, a distance metric function $D(x, y)$ is used to measure the distance between between two embeddings $f(x)$ and $f(y)$. Then gradients that either decrease or increase this distance are made depending on their semantic data and the DML method used (Schroff et al., 2015). Examples of some standard distance metrics are Euclidean distance, city block distance and cosine similarity. It is common to normalize these features when computing the similarity as it makes the comparison intuitive and efficient (Xuan et al., 2020).

3. Related Work

In the early work on font generation, methods mainly relied on shape modeling of outlines and interpolation (Suveeranont and Igarashi, 2010; Campbell and Kautz, 2014), these got good result but were easily overwhelmed when performing on font styles with more low-level handcrafted features (graphic, geometric).

With the introduction of deep neural networks (DNN) (Krizhevsky et al., 2012), it was doable to confront the low-level features. DNN is further described in Section 2.1. Upchurch et al. (2016) used DNN in order to separate the content and style representation of characters, which could then be used for transferring font styles. While this improved the state-of-the-art, there were still problems with blurry or garbled glyphs that did not match the style in all respects (Upchurch et al., 2016).

Generative adversarial network (GAN) has made a considerable impact in font generation. GAN has the potency of DNN at learning low-level handcrafted features, but also the property at making images less distinguishable from the training dataset, which results in a substantial reduction in blurry glyph or garbled glyph generation. GAN is further described in Section 2.3. Yuchen (2017) was able to get impressive results with his zi2zi model, which was built by extending the pix2pix model, which is a well-known image-to-image translation model that utilizes GAN. The Image-to-image translation and the pix2pix model is described in Section 2.8. However, the zi2zi model requires hundreds of glyphs with coherent style in order to fully work. This is in many cases unpractical. Section 3.2 describes the zi2zi model.

Few-shot font generation on the other hand relies only on a few samples to generate an entire font library. This presents the possibility of real-time practicability, and reduces the number of samples required to generate full font libraries. Azadi et al. (2018) proposed the first few-shot font generation method, Multi-conditional Generative Adversarial Network for Image Synthesis (MC-GAN). However, MC-GAN's input content is limited to 26 Latin capital letters, and attempting to scale to handle larger writing system such as Chinese will likely be impossible. Section 3.2 describes MC-GAN. One of the most recent state-of-the-art font generation models is Multiple Localized Experts (MX-Font). This model utilizes multiple encoders which attends to different local concepts for a given character. Section 3.3 describes MX-Font. Font generation is usually performed within the same language (Zhang et al., 2018b; Zhu et al., 2020; Gao et al., 2019; Park et al., 2021b). However, recently Li et al. (2021) proposed an end-to-end solution to cross language font generation. This model is called Few-Shot Font Style Transfer Between Different Languages (FTransGAN). Section 3.4 describes FTransGAN. Lastly, Section 3.5 describes a proposed framework for improving the style feature extraction of font generation models with the use of Deep Metric Learning.

The introduction to this chapter is rewritten from the specialization project mentioned in the preface.

3.1. Zi2zi

Zi2zi (Yuchen, 2017) is a follow-up project to the earlier project called Rewrite. Rewrite is a font generation model that addresses the font generation process as a style transfer problem, and makes use of deep neural network. However, there are some issues with Rewrite (Yuchen, 2017):

- The generated images are oftentimes blurry
- Fails under more stylized fonts
- Limited to learn and output only one target font style at a time

Yuchen (2017) attempts at solving these problems with the zi2zi model. The model borrows the generator and discriminator directly from the pix2pix UNET model (Isola et al., 2017). The encoder is trained at mapping a font glyph image x to an embedded space representation, further referred to as character embedding. An approach such as this one, where a higher dimensional space X is mapped into a lower dimensional space Y making an injective map $f : X- > Y$ is called low-dimensional embedding (Russell and Norvig, 2010). Low-dimensional embedding makes it possible for the encoder to discover suitable internal representation of the data, while simultaneously learning how to extract the characteristics of the character. The input to the decoder is a concatenation of the character embedding and a category embedding, which represents the style. This category embedding is a non-trainable gaussian noise z that is unique to each font. The target output of the decoder is a font glyph image y that is the same character as the character embedding and that also represents the style of the category embedding. An overview of this architecture is shown in Figure 3.1.

To achieve this, multiple losses are utilized. We define f as the encoder and g as the decoder of the generator G, which inherently gives us $G = g \circ f$. Additionally, d is a distance function such as mean squared error, $P(C|X)$ describes the probability distribution over the class labels given by D and s describes the set of font glyph images in the dataset. The objective function is:

$$G^* = \arg\min_G \max_D L_{cGAN} - \alpha L_C + \beta L_{L1} + \gamma L_{CONST} + \delta L_{TV} \tag{3.1}$$

for some weights $\alpha, \beta, \gamma, \delta$, where:

$$\begin{aligned} L_{cGAN} &= \mathbb{E}_{x,y}[\log(D(x,y))] + \mathbb{E}_{x,z}[1 - D(x, G(x,z))] \\ L_{L1} &= \mathbb{E}_{x,y,z}[||y - G(x,z)||_1] \end{aligned} \tag{3.2}$$

L_{cGAN} and L_{L1} are directly retained from the pix2pix model, which is described in Section 2.8. L_{cGAN} encourages the model at producing images that cannot be distinguished

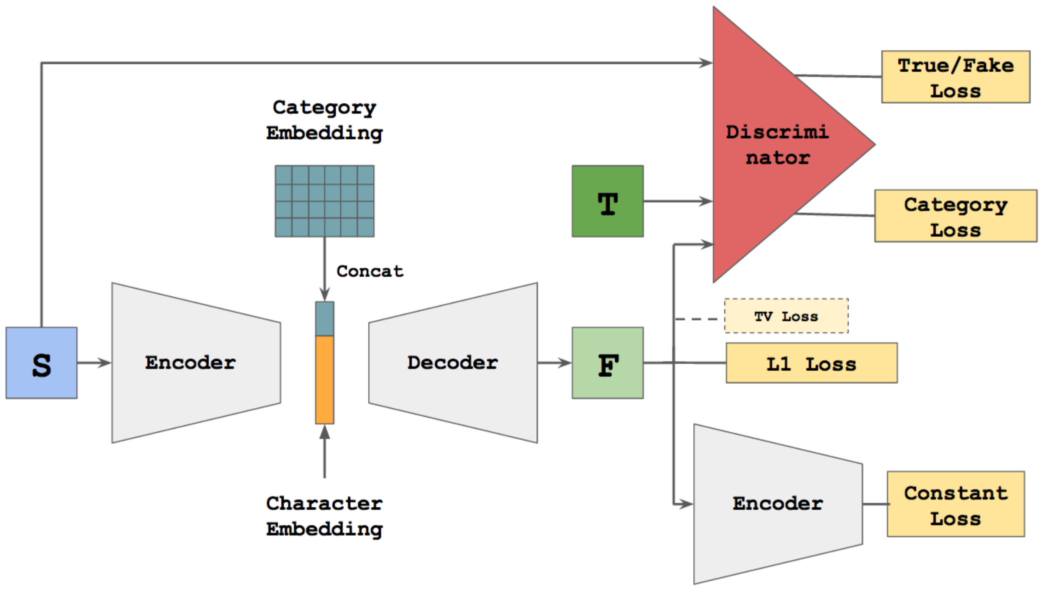


Figure 3.1.: Overview of Zi2Zi's architecture (reused with permission from Tian Yuchen)

from "real" images, therefore making the output more realistic and sharp. L_{cGAN} is described further in Section 2.3 and 2.4. L_{L1} tasks the generator at making the generated images closer to ground truth, punishing deviations by mean absolute error. While L_2 which uses mean squared error is an option, L_1 is preferred as it encourages less blurring (Isola et al., 2017).

$$L_C = \mathbb{E}_y[\log P(C = c|y)] + \mathbb{E}_{x,z}[\log P(C = c|G(x, z))] \tag{3.3}$$

L_C is derived from the AC-GAN model (Odena et al., 2017). This loss penalizes generated images where the style deviates from the provided targets, by making the discriminator predict the style of the generated characters, therefore preserving the style itself. This loss is subtracted instead of being added, as the goal for the generator and the discriminator is to maximize L_C; maximizing the chance of the discriminator recognizing the style of the generated image.

$$L_{CONST} = \sum_{x \in s} d(f(x), f(G(x, z))) \tag{3.4}$$

L_{CONST} is derived from DTN network (Taigman et al., 2016). This loss follows the idea that the character embedding of the input of G should match the character embedding of the output of G, as they are of the same character.

$$L_{TV}(\hat{y}) = \sum_{i,j} \left((\hat{y}_{i,j+1} - \hat{y}_{i,j})^2 + (\hat{y}_{i+1,j} - \hat{y}_{i,j})^2 \right)^{\frac{1}{2}} \tag{3.5}$$

L_{TV} is an anisotropic total variation loss, and is added to smooth the resulting image, removing noise and making the image overall more pleasing to look at. $\hat{y}_{i,j}$ denotes the pixel value at position $\{i, j\}$ and $\hat{y} = [\hat{y}_{i,j}] = G(x, z)$. This loss is regarded as optional by Yuchen (2017) for various reasons, one of them being that smoothing makes the resulting image less sharp.

The strategy to train the model for generating a font is done in a two-step process. First, train the model at many fonts, teaching the generator how to extract character structural information. Next, the layers of the encoder is frozen and the decoder is fine-tuned. The reason the decoder is fine-tuned (while the encoder is not) is that each style embedding is unique to every font, therefore one would no longer need the other fonts for this part of the training. The encoder, on the other hand, needs to see multiple of the same character to build a proper character embedding of the character.

The results of Zi2Zi are quite impressive. While Yuchen (2017) did not directly compare his model to the state-of-the-art, it is quite noticeable it achieved state-of-the-art performance just by the examples of his results and the recognition of his work from multiple papers (Park et al., 2021b; Li et al., 2021; Xi et al., 2020; Xie et al., 2021).

3.2. MC-GAN

Azadi et al. (2018) proposed the first end-to-end solution for few-shot font generation with Multi-conditional Generative Adversarial Network for Image Synthesis (MC-GAN). The solution is specifically designed to predict ornamented glyphs ranging from the letter A to Z. It does so using a novel stacked cGAN architecture called GlyphNet to predict glyph shapes, and a novel ornamentation network called OrnaNet to predict color and texture of the final glyphs.

The first network, the GlyphNet, generates all 26 glyphs given a few example glyphs. The size of the glyphs are 64×64, and each glyph is stacked on top of the other, resulting in the input and output dimension $B \times 26 \times 64 \times 64$. The reason that the glyphs are stacked on top of each other, instead of a basic tiling, is because that would fail to capture correlations as no convolution field would realistically be able overlap all glyphs, even at the lowest depth.

As mentioned in Section 2.4, the adversary between generator and discriminator for a regular cGAN can be formulated as:

$$L_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z)))] \tag{3.6}$$

MC-GAN uses a similar approach as loss functions for GlyphNet, but makes some necessary changes to fit the task. Firstly, as there is already natural occurrence of noise brought by the random selection of example glyphs, the z value used to generate noise is considered redundant and therefore discarded. Secondly, MC-GAN adopts the least squared loss function, changing the GAN to a LSGAN (Mao et al., 2017). A description of the GAN loss function can be found in Section 2.3. LSGAN is used instead of GAN, as it produces higher quality results and is more stable (Mao et al., 2017). Thirdly, the loss function is split into a local and global loss function. This is done to include the

PatchGAN architecture, which contributes to making the generated images sharp and realistic. The PatchGAN architecture is described further in Section 2.8. Lastly, the L_1 loss is added to penalize deviations of generated images $G_1(x_1)$ from their ground truth y_1. The loss function is,

$$\begin{aligned} L(G_1) &= \lambda L_{L_1}(G_1) + L_{LSGAN}(G_1, D_1) \\ &= \lambda \mathbb{E}_{x_1,y_1}[||y_1 - G_1(x_1)||_1] \\ &+ \mathbb{E}_{x_1,y_1]}[(D_1(y_1) - 1)^2] \\ &+ \mathbb{E}_{x_1}[D_1(G_1(x_1))^2], \end{aligned} \tag{3.7}$$

where $L_{LSGAN}(G_1, D_1) = L^{local}_{LSGAN}(G_1, D_1) + L^{global}_{LSGAN}(G_1, D_1)$.

For training, their 10K font data set is used. In each training iteration a few random subsets of y glyphs are used, while the others are whited out, as x.

The second network, OrnaNet, is trained to generate style and ornamentation to the glyphs generated from GlyphNet. All 26 glyphs are generated, including the observed glyphs, as it lets the OrnaNet generate high quality stylized letters even if the generated glyphs were to be coarse and differentiate considerably from the observed glyphs. To generate the observed glyphs, a leave-one-out approach is used. A prediction from GlyphNet is done where all but one observed glyph is used as input. Then the one observed glyph that was retracted is picked from the output and stored. This process is repeated for each observed glyph. Afterwards, these stored observed glyph predictions are combined with an output from GlyphNet where all observed glyphs are used as input. As OrnaNet predicts colored images, it uses the input and output dimension $B \times 3 \times 26 \times 64 \times 64$. For this reason, the images generated from GlyphNet must be converted from grayscale to colored as well, which is done through a reshape transformation and gray-scale channel repetition, represented by $\mathcal{T}$.

The loss function for OrnaNet is similar to GlyphNet, but a mean square error loss between binary masks of the input and output is included as well. The binary masks is obtained by passing images through a sigmoid function, indicated as σ. This is done to achieve clean sharp outlines in the color images. The loss function is,

$$\begin{aligned} L(G_2) &= L_{LSGAN}(G_2, D_2) + \lambda_1 L_{L_1}(G_2) + \lambda_2 L_{MSE}(G_2) \\ &= \mathbb{E}_{x_2,y_2}[(D_2(y_2) - 1)^2] \\ &+ \mathbb{E}_{x_2}[D_2(G_2(x_2))^2] \\ &+ \mathbb{E}_{x_2,y_2}[\lambda_1 ||y_2 - G_2(x_2)||_1 \\ &+ \lambda_2(\sigma(y_2) - \sigma(G_2(x_2)))^2], \end{aligned} \tag{3.8}$$

where $x_2 = \mathcal{T}(G_1(x_1))$ and $L_{LSGAN}(G_2, D_2) = L^{local}_{LSGAN}(G_2, D_2) + L^{global}_{LSGAN}(G_2, D_2)$.

To evaluate the model, Azadi et al. (2018) compared MC-GAN to the patch based synthesis method by Yang et al. (2017). Overall users preferred the prediction from MC-GAN 80.0% of the time.

While the results of MC-GAN are impressive, it still has some limitations for practical usage; the input content is limited to 26 Latin capital letters, and attempting to scale this to handle larger writing system such as Chinese will likely be impossible. Additionally, the amount of parameters are extremely large as all letters are passed in one forward pass. This increases the storage requirements drastically.

3.3. MX-Font

One of the more recent font generation models is Multiple Localized Experts Few-shot Font Generation Network (MX-Font) (Park et al., 2021b). MX-Font utilizes a stack of encoders called multiple localized experts where each expert attends to different local concepts for the given character. The experts each compute a local content feature and local style feature which are thereafter combined to form the generated glyph. MX-font uses a component and a style feature classifier, a generator and a discriminator, in addition to the multiple localized experts. An overview of the architecture is shown in Figure 3.2.

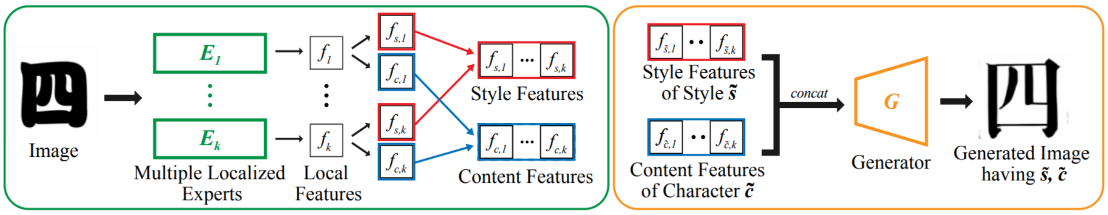


Image
E_1
E_k
Multiple Localized Experts
f_1
f_k
Local Features
Style Features
Content Features
Style Features of Style $\tilde{s}$
Content Features of Character $\tilde{c}$
concat
G
Generator
Generated Image having $\tilde{s}, \tilde{c}$

Figure 3.2.: **Overview of MX-Font (reused with permission from Song Park).** The experts E_i encode the input image to local features f_i (green box). The local content feature f_{c_i} and local style feature f_{s_i} are then computed from f_i. Afterwards, the features are combined and passed to the generator G to generate the target image (yellow box).

Chinese characters can be described as a combination of components. One example is ' 智', that can be decomposed to the components ' 矢', ' 口' and ' 日'. The reason MX-Font utilizes several experts is to take advantage of this decomposition; by labeling each character with a component label U_c, which contains the components of the character, and by delegating the task of predicting the features of the components to separate encoders. The experts E_i each produce $f_{s,i}$ and $f_{c,i}$ by first generating a feature $f_i = E_i(x) \in \mathbb{R}^{d \times w \times h}$ where d is a feature dimension and $\{w, h\}$ are the spatial dimension. Then multiplying f_i with two linear weights $W_{i,c}, W_{i,s} \in \mathbb{R}^{d \times d}$, a local content feature $f_{c,i} = W_{i,c}^{\top} f_i$ and a local style feature $f_{s,i} = W_{i,s}^{\top} f_i$ are computed (Park et al., 2021b).

The delegation is done by the component classifier CLs_u and style classifier CLs_s. The component classifier CLs_u produces a prediction probability $p_i = CLs_u(f_{c,i})$, where $p_i = [p_{i0}, ..., p_{im}]$, p_{ij} is the confidence scalar value of the component j and m the number of components in U_c. These confidence scalar values are then used to map experts E_i to different components u_j. The goal for expert-component matching is to find the set of confidence scalar values that fulfill the criteria:

- The number of total allocations is $max(k, m)$

- Each expert and component are allocated
- The sum of predictions is maximized

An example of a delegation is shown in Figure 3.3. This expert-component matching is reformulated as a Weighted Bipartite B-Matching problem, and solved using the Hungarian algorithm (Kuhn, 1955). More details about the problem reformulation can be found in the MX-Font paper (Park et al., 2021b).

The style classifier Cls_s predicts the style label by $Cls_s(f_{s,i})$. As the target style labels are the same for each expert, no matching problem is necessary for this classifier.

For training, $n = 3$ glyphs that share the same content label c (but random styles) and n glyphs that share the same style label s (but random content) are fetched. The generator then generates a glyph with the same content label as c and the same style label as s. This is process is done in parallel for 8 different glyphs, creating a mini batch size of 24.

The full objective function of MX-Font is:

$$\begin{aligned} L_D &= L_{adv}^D, \\ L_G &= L_{adv}^G + \lambda_{L1} L_{L1} + L_{fm} \\ L_{exp} &= \sum_{i=1}^{k} [L_{s,i} + L_{c,i} + L_{indp,i} + L_{indpexp,i}] \end{aligned} \tag{3.9}$$

L_{adv}^D and L_{adv}^G are the hinge generative adversarial losses, which are described in Section 2.5. L_{L1} tasks the generator at making the generated images closer to ground truth, punishing deviations by mean absolute error:

$$L_{L1} = \mathbb{E}_{x,y}[||y - G(x)||_1] \tag{3.10}$$

where $x = \{c, s\}$ and y is the target image. $\mathcal{L}_{fm}$ is the feature matching loss, formulated as follows:

$$L_{fm} = \mathbb{E}_{x,y}[\sum_{l=1}^{L-1} ||D^l(y) - D^l(G(x))||_1] \tag{3.11}$$

where L is the number of layers in the discriminator D and D^l denotes the output of l-th layer of D. The losses $L_{s,i}$ and $L_{c,i}$ utilize cross entropy loss to train the experts at predicting the correct style and component labels respectively, and attempt at maximizing the entropy of the outputs of the classifiers given the same input, to make the features disentangled and not correlated. $L_{indp,i}$ and $L_{indpexp,i}$ use Hilbert-Schmidt Independence Criterion (Gretton et al., 2005) of style features, content features and local features to make certain they are independent from another. More detail about L_{exp} can be found in the MX-Font paper (Park et al., 2021b).

Because Korean characters can be decomposed to components in a similar way as Chinese characters, Park et al. (2021b) were able to compare their model both on Chinese characters and cross language, using Chinese characters for style reference and Korean characters for component reference. Using only Chinese characters, MX-Font received less score on LPIPS (Zhang et al., 2018a) and Fréchet inception distance (FID) (Heusel et al., 2017), than their previous model LF-Font (Park et al., 2021a). LPIPS measures perceptual dissimilarity between the ground truth and the generated glyphs, while FID measures the similarity of generated images to real ones using the Inception-v3 model. Accuracy and user score were divided on scores based on content, style and both. MX-Font scored the highest on all accuracy and user scores other than the user score based on context. For cross language prediction MX-Font outperformed all other models on LPIPS, FID, all accuracy scores and all user scores.

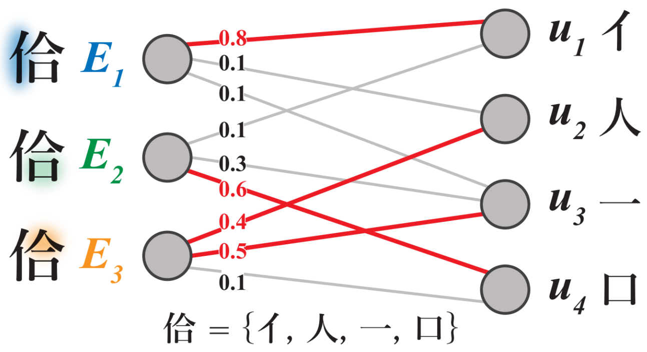


Figure 3.3.: **An example of localized experts (reused with permission from Song Park).** The number of experts is three (E_1, E_2, E_3) and the number of target components is four (u_1,...,u_4). The red edges illustrate the optimal solution for expert-component matching.

3.4. FTransGAN

Few-Shot Font Style Transfer Between Different Languages (FTransGAN) is a font generation method proposed by Li et al. (2021) that is the first to apply an end-to-end solution to cross language font generation. This method introduces two novel modules, Context-aware Attention Network and Layer Attention Network that capture both local and global style features simultaneously through usage of the attention mechanism.

This model uses a generator G and two discriminators $D_{content}$ and D_{style}. The generator takes in the inputs c and s, where c is a glyph image with a standard style (e.g., Microsoft YaHei) and the same content as the target image, and s is a set of glyph images $s = \{s_1, ...s_k\}$ all with the same style and different content as the target image y. The task of the generator is to extract a content feature representation from c, and a style feature representation from s. The generator consists of a content encoder, style encoder and decoder. The content encoder extracts the content representation from the given font glyph image, while the style encoder extracts the style representation from the given font glyph images. The content encoder architecture consists of three convolutional layers. The style encoder architecture reuses the content encoder architecture, and adds two convolutional layers, three Context-aware Attention Network and a Layer Attention Network.

The Context-aware Attention Network are utilized at three receptive fields, 13x13, 21x21, 37x37. The shallower layers see local features, while the deeper layers can see almost the entire image. The input is a feature map with a size of $C \times H \times W$ given by the last convolutional layer, where C, H, W denote the number of channels, height and width respectively. Each region of the feature map is denoted as $\{v_r, r = 1, 2, ..., H \times W\}$ (Li et al., 2021). A Self Attention layer generate the context information of the feature map from each region by:

$$h_r = SA(v_r). \tag{3.12}$$

This is then further incorporated to a latent representation with:

$$u_r = \tanh(W_c h_r + b_c). \tag{3.13}$$

Here, u_r describes the context information by latent variables. This context information is not limited to the receptive fields of the layer but also the contextual information from other regions. A context vector u_c is trained with the entire model and employed to assign each region an attention score based on the context information. This is done as all regions are not believed to have the same contribution. These scores are then normalized by a softmax layer and used in a weighted sum to obtain a feature vector f. Specifically,

$$a_r = \text{softmax}(u_r^T u_c), \tag{3.14}$$

$$f = \sum_{H \times W} a_r v_r. \tag{3.15}$$

As there are three parallel Context-aware Attention Networks, f_1, f_2, f_3 are obtained.

The Layer Attention Network takes in four inputs; feature mask f_m given by the last convolutional layer, and the feature vectors f_1, f_2, f_3. An one-layer neural network is utilized to assign each feature vector a score depending on the feature mask. This score determines how much each Context-aware Attention Network contributes to the style representation. Specifically,

$$w_1, w_2, w_3 = \text{softmax}(\tanh(W_1 f_m + b_l)), \tag{3.16}$$

$$z_j = \sum_{i=1}^{3} w_i f_i, \tag{3.17}$$

where w_1, w_2, w_3 are three normalized scores given by a neural network, z is the weighted sum of three feature vectors, and z_j denotes the style representation of the j-th style image. As the style encoder accepts K style images, the final latent code z is the mean of all vectors z_j:

$$z = \frac{1}{K} \sum_{j=1}^{K} z_j. \tag{3.18}$$

This vector z is C-dimensional vector, but the content representation is $C \times H \times W$, therefore it is copied $H \times W$ times to match the size of the content representation. The output from this operation is the style representation.

The discriminators have almost the same architecture; only the input dimension differs. $D_{content}$ receives a concatenation of the content image and the generated image, and checks whether they are the same character. D_{style} receives a concatenation of the style images and the generated images, and checks whether they are of the same style. The PatchGAN architecture is utilized for the discriminator, as it contributes to making the generated images sharp and realistic. The PatchGAN architecture is described further in Section 2.8.

The objective function for FTransGAN is:

$$L = \lambda_{L1} L_{L1} + \lambda_s L_{style} + \lambda_c L_{content} \tag{3.19}$$

where λ_{L1}, λ_s, λ_c are three weights for balancing these terms. For higher quality results and to stabilize GAN training, both $L_{content}$ and L_{style} use hinge generative adversarial loss as opposed to the standard cGAN loss. Section 2.5 describes the hinge loss and Section 2.3-2.4 described the standard cGAN loss. L_1 loss tasks the generator at making the images closer to the ground truth, punishing deviations by mean absolute error:

$$L_{L1} = \mathbb{E}_{x,y}[||y - G(x)||_1], \tag{3.20}$$

where $x = \{c, s\}$ and y is the target image. While this font generation model was the first to apply an end-to-end solution to cross language font generation, it was still able to compare its results to two other models EMD (Zhang et al., 2018b) and DFS (Zhu et al., 2020), as these models could be modified for cross language font generation. FTransGAN was able to achieve state-of-the-art results compared to the other models, and in a user preference survey the users preferred their model 80.3% of the times.

3.5. Few-Shot Font Generation with Deep Metric Learning

Aoki et al. (2021) proposed a framework for few-shot font style transfer for extracting better style features using Deep Metric Learning (DML). A description of DML can be found in Section 2.9.

The framework is a network which consists of 3 layers: L_2 normalization, fully connected layer and softmax. The framework is implemented as a parallel network alongside a few-shot font generation model that retrieves the style features as input during training and tries to predict the class label. The loss in this prediction is then backpropagated to the style encoder, which encourages the style encoder to make the class of each style features easier to classify. The DML method used is the L_2-constrained softmax loss proposed by Ranjan et al. (2017) because of its robustness and learning stability. The L_2-constrained softmax loss for their framework is expressed as:

$$\mathcal{L}_{dml}(x_i) = -\log \frac{\exp{(W_{c_i}^\top x_i + b_{c_i})/\tau}}{\sum_j \exp{(W_j^\top x_i + b_j)/\tau}} \tag{3.21}$$

where x_i is the L_2-normalized style feature embedding for the i-th sample. c_i is the font class label corresponding to x_i. W and b are the weights and bias for the fully connected layer.

To verify the effectiveness of this framework, Aoki et al. (2021) tested it on AGIS-Net (Gao et al., 2019) and EMD (Zhang et al., 2018b). The test results showed that the framework is able to improve the results for both models, especially for cases where the style reference glyphs are considerably limited.

4. Method

In this chapter, the novel few-shot font style transfer with Extraction of Partial Style (EPS-Font) is presented. Section 4.1 gives an overview of EPS-Font. Sections 4.2-4.3 describe the architectural choices, that relate to **RQ1**. Section 4.4 describes the objective function for training the method. Section 4.5 describes how to apply Deep Metric Learning to the method, and answers **RQ2**. Section 4.6 discusses a modification of EPS-Font made to answer **RQ3**. This modification is referred to as Deformation and Texture Separation (DTS). Figure 4.2 and Figure 4.3 are reproduced from the report in TDT12 mentioned in the preface.

4.1. Model

The task of a few-shot font style transfer is to predict the target image x, given the content image c and a few reference style images s. For font style transfer, the content corresponds to the character of the glyph image, while the style corresponds to the font of the glyph image. The content image should be of the same content as x, but be of a standard style, for instance Microsoft Yahei. The style images s are a set of images $\{s^1, s^2 ... s^K\}$ which share the same style as x, but are of different content.

Similar to the most recent font style transfer methods, this model disentangles the content and style features from glyph images (Gao et al., 2019; Park et al., 2021b; Li et al., 2021; Xie et al., 2021). This will be the job of the generator G, which consists of a content encoder, a style encoder and a decoder. The encoders extract the content and style representation, respectively. The combination of these representations will then be fed to a decoder, which generates the target image. Two discriminators, content discriminator and style discriminator, will be utilized to train the generator G. Since multiple encoders with different roles are utilized by the generator G, multiple discriminators are beneficial to validate the effect each encoder has on the generated image. The decoder and discriminators are borrowed from Li et al. (2021)'s few shot style transfer between different languages (FTransGAN). FTransGAN is described in Section 3.4.

4.2. Generator

The content encoder consists of three convolutional layers, each followed by batch normalization (BN) and Rectified Linear Unit (ReLU). BN and ReLU are described in Section 2.1 and Section 2.1, respectively. The style encoder consists of six convolutional layers, likewise followed by BN and ReLU, and an upsample layer. The style encoder has deliberately more convolutional layers than the content encoder. This is done for three reasons:

- The amount of style images in a dataset is usually very large. For instance, 3.1M glyph images are used in the dataset used by Park et al. (2021b) in their paper proposing Multiple Localized Experts Few-shot Font Generation Network (MX-Font). While the amount of content images is usually small in comparison. In the experiments conducted in Chapter 5, approximately 1000 glyph images are used to represent content. As the style encoder has the task of differentiating and representing more images than the content encoder, it is fitting that the style encoder has more convolutional layers to make up for this difference.
- For typical style transfer, the style is defined as a set of texture and color, and it is typical to use the same amount, or one more, of convolutional layers for the style encoder as the content encoder (Huang and Belongie, 2017; Gatys et al., 2016; Johnson et al., 2016; Chandran et al., 2021). However, for font style transfer the style also consists of deformation, which is referred to as the difference of shape (e.g. cursive, big/small size, artistic traits), which must also be transferred.
- The style information is globally present for typical style transfer, while for font style transfer, it is only partially present, as it is tied to the content. For instance, the style is less prominent in the image shown in Figure 4.1a than the style image shown in Figure 4.1b. Therefore, the style images serve as *partial* style reference.

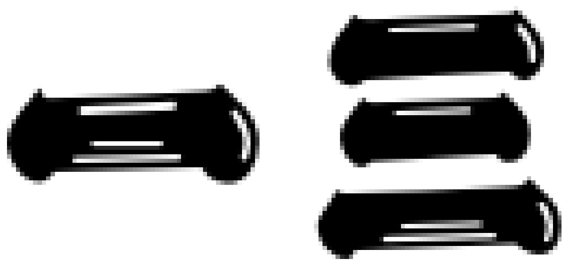

(a) Style image of " 一" (b) Style image of " 三"

Figure 4.1.: Style images example.

The final convolutional layer of the style encoder reduces the spatial dimension of the input from 4×4 to 1×1. This is essentially used to obtain all features, and combine them to make a style feature which characterizes the font. This output is then upsampled to the spatial dimension $w \times h$, where $\{w, h\}$ is the spatial dimension of the content features, to cover the content feature. These two final layers alleviate the issue of style images only serving as partial style reference, as the upsampling distributes the style information evenly throughout the spatial dimension. An overview of this architecture is shown in Figure 4.2. As you can notice in the architecture, an alteration is applied to the network when L_{SC} is used. This will be described in Section 4.5.

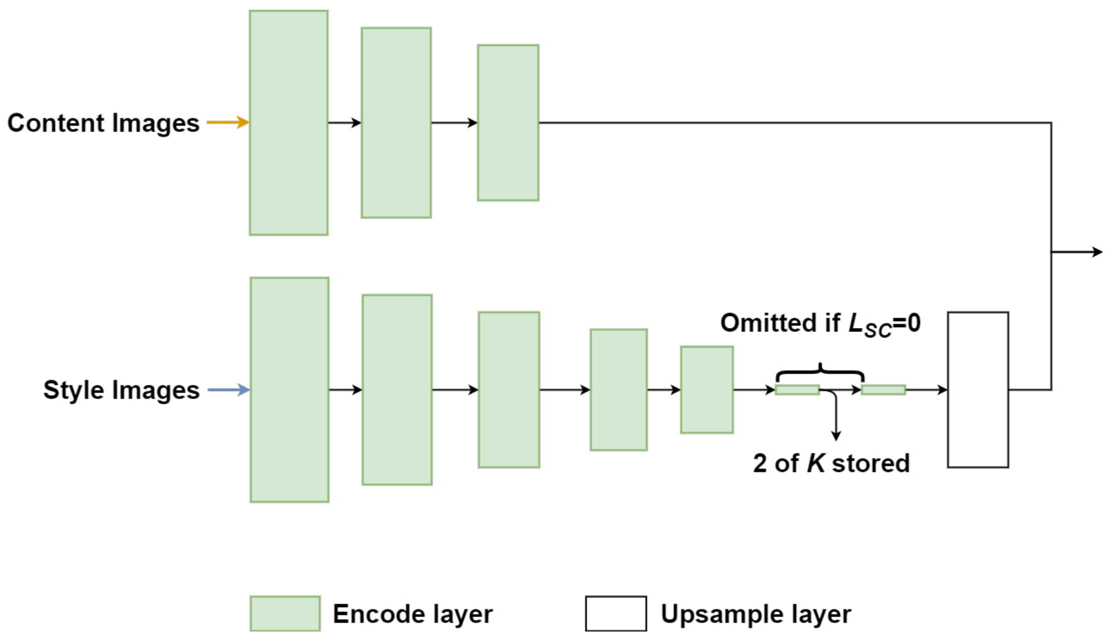


Figure 4.2.: **Architecture of the encoders of EPS-Font.** An encode layer consists of a convolutional layer followed by BN and ReLU. Two of the K style images' embeddings are stored for the calculation of L_{SC}.

The content features and the combined style features are then combined and decoded through a six residual network block, and two transposed convolutional layers, each followed by BN and ReLU. A residual network block is a convolutional layer $\mathcal{F}(x)$ with a "shortcut connection" which forwards the input x. The output of the residual network block is $\mathcal{F}(x) + x$. Residual network blocks are shown to be easier to optimize and give high accuracy from considerably increased depth (He et al., 2016). Transposed convolutional are convolutional layers which upsample the spatial dimensions. Finally, the output of the final residual network block goes through a convolutional layer and tanh layer, outputting the generated image. An overview of this architecture is shown in Figure 4.3.

4.3. Discriminator

The discriminators use the PatchGAN architecture, as the PatchGAN architecture contributes to making the generated images sharp and realistic and has shown great results for previous font style transfer methods (Azadi et al., 2018; Li et al., 2021). The PatchGAN architecture is described in Section 2.8.

As mentioned in Section 4.1, the proposed EPS-Font architecture makes use of multiple discriminators to verify the effects of the encoders. The content discriminator receives a combination of the content image c and the generated image $\hat{y}$. If their contents do not match, the content discriminator detects this and gives corresponding losses.

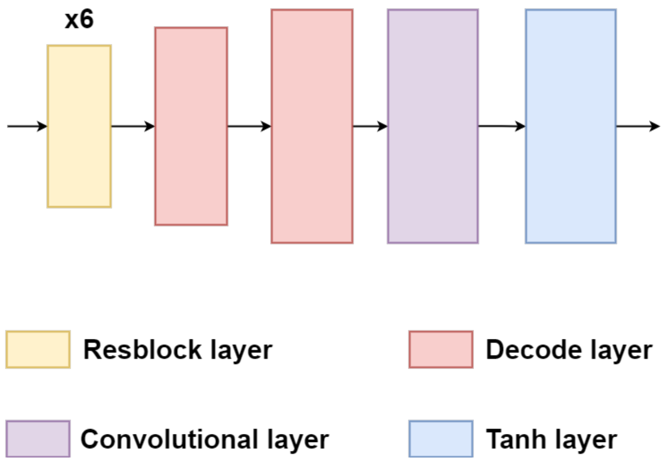


Figure 4.3.: **Architecture of the decoder of EPS-Font.** A decode layer consists of a transpose convolutional layer followed by BN and ReLU.

The style discriminator receives a combination of the style images s and the generated image $\hat{y}$. If their styles do not match, the style discriminator detects this and gives corresponding losses. Additionally, unrealistic and blurry images will be punished by both discriminators.

4.4. Objective Function

The objective function is:

$$L = \lambda_{L1} L_{L1} + \lambda_s L_{style} + \lambda_c L_{content} + \lambda_{SC} L_{SC}, \tag{4.1}$$

where λ_{L1}, λ_s, λ_c. λ_{SC} are weights for balancing these terms. For higher quality results and to stabilize GAN training, both $L_{content}$ and L_{style} use hinge loss versions of the cGAN loss. Section 2.5 describes hinge loss. The cGAN loss is described in Sections 2.3-2.4. L_1 loss tasks the generator at making the images closer to the ground truth:

$$L_1 = \mathbb{E}_{\hat{x},x \sim P(\hat{x},x)}[||x - \hat{x}||_1]. \tag{4.2}$$

L_{SC} is the Selectively Contrastive Triplet loss proposed by Xuan et al. (2020), which is described in Section 4.5.

4.5. Deep Metric Learning

As mentioned in Section 1.2, it is interesting to study if Deep Metric Learning (DML) can be beneficial to few-shot font style transfer methods. A description of DML is available in Section 2.9. To test the effectiveness of DML a method proposed by Xuan et al. (2020) is used, which is a modification to the triplet loss.

Triplet loss is a common approach for DML, proposed by FaceNet (Schroff et al., 2015). Triplet loss penalizes the relative similarity between three examples: x_a, x_p and x_n, where x_a and x_p are of the same class and x_n is of a different class. The goal is to minimize the distance between x_a and x_p, and to maximize the distance between x_a and x_n. The formula for triplet loss is (Hermans et al., 2017):

$$L_{tri} = \sum_{\substack{a,p,n \\ c_a = c_p \neq c_n}} [m + D(x_a, x_p) - D(x_a, x_n)], \tag{4.3}$$

where $D(x, y)$ is a distance metric function that measures the distance between $f(x)$ and $f(y)$. This loss enforces that the distance between the positive pair (x_a, x_p) (of the same class c_a) are smaller than the distance of the negative pair (x_a, x_n) (of a different class c_n by at least margin m). An example of the triplet loss is shown in Figure 4.4.

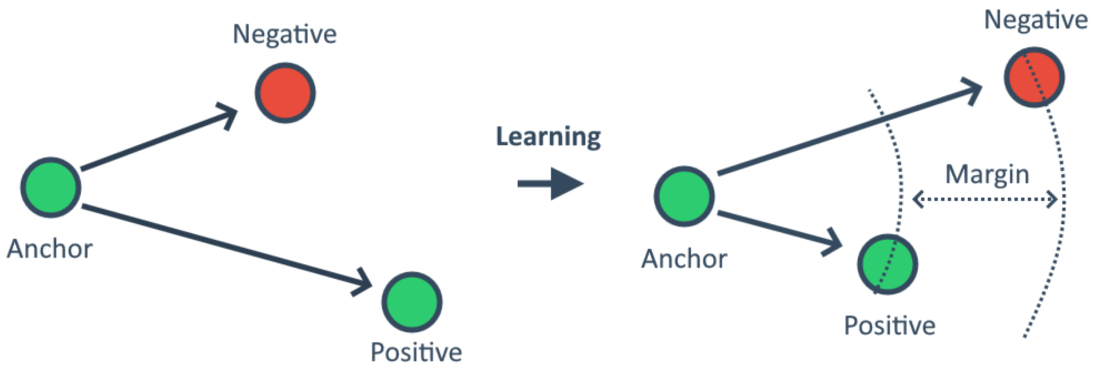


Figure 4.4.: Triplet loss example. (Reproduced with permission from Florian Schroff.)

A major caveat of the triplet loss, though, is that as the dataset gets larger, the possible number of triplets grows cubically, rendering a long enough training impractical (Hermans et al., 2017). Additionally, as more training is performed on f, the more triplets fulfil the margin criteria, making those triplet losses to be equal to zero.

This makes effective selections of triplets during training very important. For instance, one could only select x_n such that $\arg\min_{x_n} D(x_a, x_n)$, in other words the hardest examples for the model. This selection strategy is called *hard* negative mining. However, a variety of work shows that training on only the hard negative examples leads to bad local minima (Schroff et al., 2015; Faghri et al., 2018; Song et al., 2016). Therefore, one could use *semi-hard* negative examples instead (Schroff et al., 2015). Semi-hard negative examples are x_n where $D(x_a, x_p) < D(x_a, x_n)$ and $L > 0$. These examples are further away from the anchor than the positive example, but are still hard as the triplet does not yet fulfil the margin criteria.

Training with semi-hard triplets remedies the problem of bad local minima, but hard negative examples are the cases where the model fails to capture semantic similarity, so intuitively one would want to include these in the training in some way if possible. Xuan et al. (2020) mention two reasons as to why triplets are hard to optimize. The first problem is that the more similar $f(x_a)$ is to $f(x_n)$ and $f(x_p)$, the more effect of the triplet loss' gradients is lost during normalization. The second problem is that the more similar $f(x_a)$ is to $f(x_n)$, the more likely it is for $f(x_a)$ to pull $f(x_n)$ with it when $f(x_a)$ is modified to be pulled toward $f(x_p)$. More detail on why these problems occur can be found in the paper by Xuan et al. (2020). Therefore, Xuan et al. suggest a modification called Selectively Contrastive Triplet loss, that makes training with hard negative examples feasible. Xuan et al. perform modification to the triplet loss as follows:

$$L_{SC} = \begin{cases} -\alpha D(x_a, x_n), & \text{if } D(x_a, x_n) < D(x_a, x_p) \\ L_{tri}, & \text{otherwise.} \end{cases} \tag{4.4}$$

This modification changes the loss function so that the anchor positive pairs in triplets are not updated when the triplet pair is too tightly clustered (when $D(x_a, x_n) < D(x_a, x_p)$ applies), and instead 'focuses' on pushing apart the hard negative examples. α is a weight that decides how much of this modification is to be applied, where if $\alpha = 0$ the loss function would be a standard semi-hard negative mining.

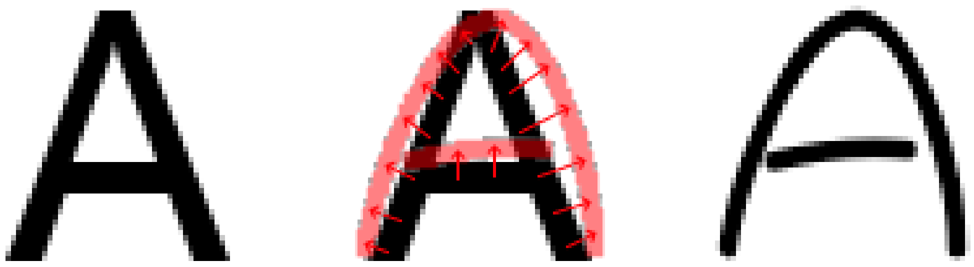
To apply L_{SC} to the architecture an extra convolutional layer, followed by BN and ReLU is added before the last convolutional layer. The reason this convolutional layer is included is to use embeddings from a layer that is both earlier than the last convolutional layer and where the embeddings have the spatial dimension $\{1 \times 1\}$, which the architecture does not have otherwise. The reason to use embeddings from a layer earlier than the last convolutional layer is because it is expected of L_{L1}, L_{content} and L_{style} to pull style images that are of the same font to one point in the embedded space. If they did not do this, the decoder would constantly be exposed to a lot of variation in the style features, which would make it harder for the decoder to learn the style. Therefore, it can be assumed that the embedded space of the last convolutional will not live on a manifold. However, the ideal condition for L_{SC} is for the style features to live on a manifold (Schroff et al., 2015; Xuan et al., 2020). Thus, the embeddings for the L_{SC} should be extracted earlier than the last convolutional layer, as these embeddings can live on a manifold.

The reason to use the embeddings from a layer with the spatial dimension $\{1 \times 1\}$ is because if embeddings from a layer with higher spatial dimensions (e.g., $\{4 \times 4\}$) is used, the embedding dimensionality would be $c * 4 * 4$, where c is the feature dimension of the layer used. This would limit the value of c drastically. For instance, Schroff et al. (2015) recommend using 128 dimensions for the pairwise check, which would in this scenario make $c = \frac{128}{4*4} = 8$, which is an insufficiently low feature dimension for a layer that deep. As mentioned in Section 2.2, deeper layers should have higher feature dimensions to learn advanced shapes.

4.6. Deformation and Texture Separation

While the content image and the target image are of the same character, most frequently the shapes between the glyphs are quite different. The target image can be bigger/smaller, cursive, or have artistic traits. This geometric variance is a key challenge with CNNs. The limitation originates from the fixed geometric structures of CNN modules: a convolution unit samples the input feature map at fixed locations; a pooling layer reduces the spatial resolution at a fixed ratio; a RoI (region-of-interest) pooling layer separates a RoI into fixed spatial bins, etc. (Dai et al., 2017).

An example of the deformation is shown in Figure 4.5. The red arrows in Figure 4.5b describe offsets that can be used to deform Figure 4.5a to Figure 4.5c. We distinguish between d_{offset} and d. An example of d_{offset} is the red arrows in Figure 4.5b, and is represented by $v \times h \times w$, where v is a vector describing the offset and $\{h, w\}$ are the spatial dimensions. An example of d is the glyph in Figure 4.5c, and is represented by $b \times h \times w$, where b is a single number to represent brightness. An important property of d_{offset} is that the positions of the offsets are given in the content image, which is a part of the input. Therefore, only the values of the offset will need to be predicted, which makes them bypass the previously mentioned key challenge with CNNs.

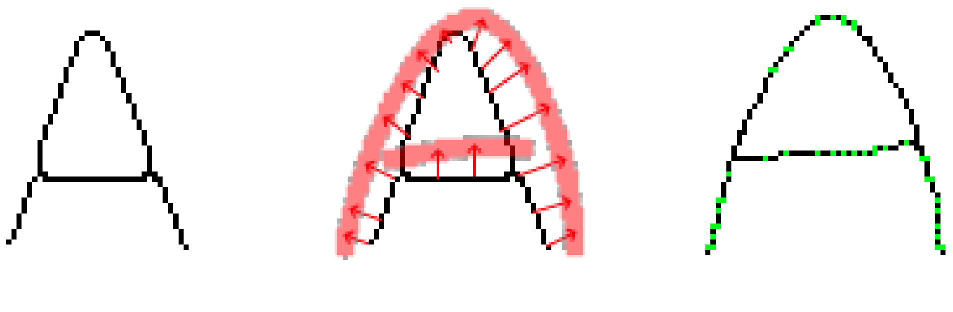


(a) Content image (b) Deformation (c) Target image

Figure 4.5.: Deformation example.

The stroke of the content image does not necessarily line up with the target image. This is solved by using a skeleton image c_{sk} generated from the content image instead. These skeleton images are generated by iteratively eroding the content image until only the skeleton remains. An example of a skeleton image is shown in Figure 4.6a. Let $G = (V, E)$ be a graph to represent c_{sk}. Vertices $v \in V$ represent all black pixels in c_{sk}. Edges $(u, v) \in E$ connect neighbor vertices. Typically, a neighbor vertex would be defined as a vertex which is adjacent vertically, horizontally or diagonally to another vertex. However, here a special condition must be applied, which is that a vertex that is adjacent diagonally is only considered a neighbor if no other vertices are adjacent to both vertices. Figure 4.7 shows why this condition is necessary and shows a way of using edge count to identify parts of a character. The stroke can be set to c_{sk} by adding surrounding black pixels to every $v \in V$ after deforming it.

Figure 4.6 contains an example where the character is stretched out slightly after being deformed. This causes gaps to form between some vertices $v \in V$. As a countermeasure, extra vertices are added in these gaps before stroke is applied.



(a) Skeleton image (b) Deformation (c) Target image[1]

Figure 4.6.: Deformation example with skeleton. Notice how the skeleton is stretched out slightly by this deformation. The green vertices in the right figure represent the extra vertices that would be added as a countermeasure for the stretching.

[1]Before stroke is applied

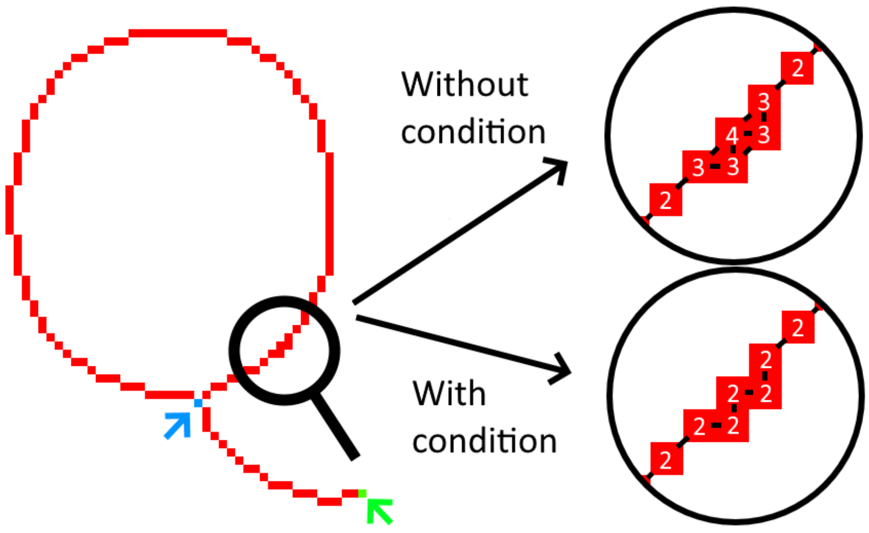


Figure 4.7.: Example of edge counts in the skeleton image. The green vertex represents a vertex with one neighbor, red vertices represent vertices with two neighbors and the blue vertex represents a vertex with three or more neighbors. For the right side instead of representing the edge count by color, a number is displayed in each vertex. It would be appropriate for these vertices to have an edge count of two, as they represent a curved line. This is cleared up by the condition that is applied to the neighbor definition.

4.6.1. Finding the deformation

The deformation itself is viewed as a free optimization problem (FOP) (Eiben and Smith, 2015). Let $p \in P$ describe each black pixel in the target image y, $c.stroke$ an estimated value of c's stroke, and $v.move$ a vector describing v's change of position that is to be performed at the end of an optimization step. The stroke is estimated using an approach similar to Newton's method. This stroke estimation approach is shown in the pseudocode in Appendix A. We denote the score as $s(d) = f_o(d) - \lambda f_p(d)$, for some weight λ. A objective function f_o reports the similarity between the deformed image d to its target image y. A penalty function f_p reports the sum of offsets for $v \in V$. The goal of the FOP is to maximize the score s. We define $\mathcal{T}$ as the mapping $\mathcal{T}(c_{sk}, y) = d$ of this FOP. For f_o, F-score with a harmonic mean of the precision and recall, F_1, is used:

$$\begin{aligned} \mathrm{PPV} &= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \\ \mathrm{TPR} &= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}, \\ f_o = F_1 &= 2 * \frac{\mathrm{PPV} * \mathrm{TPR}}{\mathrm{PPV}+\mathrm{TPR}} \end{aligned} \tag{4.5}$$

where TPR is the recall and PPV is the precision. TP, TN, FP and FN are true positive, true negative, false positive and false negative, respectively. Figure 4.8 shows examples of TP, TN, FP and FN. F_1 is used as opposed to accuracy, as TP is valued while TN is not. To illustrate why: If Figure 4.8b used an estimated stroke of 0, it would only consist of green, blue and white pixels, in other words only FN and TN. This would give a pretty high accuracy, as the number of TN is high. However, there is no similarity between the deformed image d and the target image y, as the deformed image d would be an empty image in this scenario. Therefore, the score should be 0. F_1, however, measures the harmonic mean between precision and recall, which both are 0 when TP $= 0$. Therefore, F_1 would give a score of 0 in this situation, which is the desired score.

f_p is calculated the following way:

$$f_p = \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{||\Delta(v_i, v_{i,j}) - \Delta_{start}(v_i, v_{i,j})||}{N * M} \tag{4.6}$$

where $\Delta(x, y)$ describes the relative difference in position between two vertices, $\Delta_{start}(x, y)$ describes the relative difference in position between two vertices before deformation, v_i is the i-th vertex of V, $v_{i,j}$ is the j-th vertex neighbor of v_i, N is the number of vertices in V, M is the number of neighbors of v_i. Punishing offsets by f_p encourages the deformation to also maintain the shape of the character, and not only greedily find offsets that maximize f_o.

To increase $f_o(d)$, a search around each $v \in V$ is done to find nearby FN pixels. The search radius distance is denoted as α_{dist}, and represents how far beyond $c.stroke$ the search is performed. For instance, $c.stroke = 4$ and $\alpha_{\text{dist}} = 8$ means that the search considers each p that is between 4 and 12 distance from v. Each FN pixel found results in

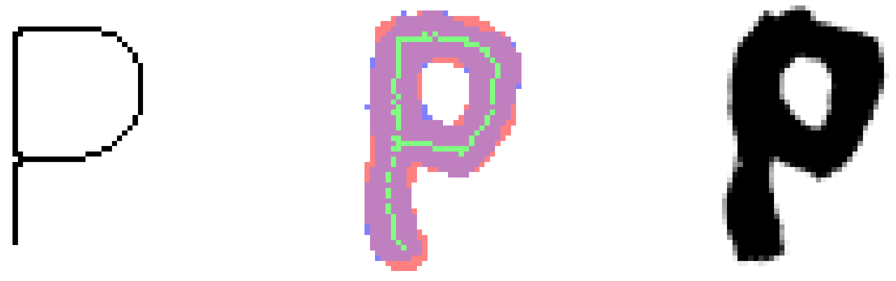

(a) Content image (b) Deformation visualization (c) Target image

Figure 4.8.: Deformation visualization example. The green pixels show d, the deformation of c_{sk}. The purple pixels represent TP, where the estimated stroke and $p \in P$ overlap. The red pixels represent FP, incorrect guessed pixels within the estimated stroke. The blue pixels represent FN, pixels which should have been within the estimated stroke. The white pixels represent TN, pixels which are not within the estimated stroke and are neither in P.

a vector with direction from v to p and size equal to $1 - (distance - 1)/(c.stroke + \alpha_{\text{dist}})$ being added to $v.move$. This will be referred to as the search method.

To decrease $f_p(d)$, the relative position and angle between each $v \in V$ are utilized. At the start of the method, the difference in position between each neighbor is saved. At each step, a new check of the difference in position is performed, how much this value has changed from its start value is then multiplied with a weight β_{pos} and subtracted from $v.move$. This simulates the characteristic of material elasticity, higher stretch receiving higher penalty. Additionally, how the angle between neighbors has changed is used as well. We denote the angle as the normalized value of the difference in position between two vertices. The difference in angle is then multiplied with a weight β_{angle} and subtracted from $v.move$. This amount is spread to neighbor vertices to even out the effect, the neighbors closest to the bend receiving the highest effects. This is to simulate the characteristic of materials bending. This will be further referred to as the stiffness method.

Let n_{steps} denote the amount of iterations performed by this method. Because the search method is more computationally expensive than the stiffness method, the search method is not performed at every step, but every α_{step}-th step instead.

Using random starting values for optimization problems are typically quite beneficial for problems with local minima (Gilli and Schumann, 2010). Unfortunately, the size of a typical font dataset for font generation is quite large. For instance, 3.1M glyph images is used at the dataset used by Park et al. (2021b) in their paper proposing Multiple Localized Experts Few-shot Font Generation Network (MX-Font). This makes the running time an important issue as well. Since each individual vertex can have a different starting value, realistically a lot of random starting values would need to be checked to be able to make benefit of them. Therefore, a deterministic starting value is used instead. The starting value is calculated in a manner that c_{sk} is resized through offset values, so that the bounding box of c_{sk} matches the bounding box of y. Intuitively, if the bounding boxes of the glyph images matches, the more similarity there is between them, therefore increasing f_o.

Some popular methods to solve optimization problems are evolutionary algorithms or branch and bound. These methods are less vulnerable to bad local minima, as they enumerate a set of candidate solutions, as opposed to only one candidate solution as in this approach. However, in Section 5.3.2, the run time to generate the font dataset with only one candidate solution is shown to be already quite high. This suggests that making use of candidate solutions (in terms of evolutionary algorithms, or branch and bound) can make the run time far too excessive. Only with smaller datasets should one consider these methods.

4.6.2. Applying DTS to few-shot font style transfer

The explanations so far have described how to find a deformation d of the content image c that can be used to represent the shape of x. But how can this be applied to EPS-Font?

As mentioned earlier, the task of a few-shot font style transfer model is to predict the target image y given the content image c and the style images s. This task can therefore be formulated as $P(y|c, s)$. This modification can be applied to a few-shot font style transfer model by splitting the task of $P(y|c, s)$ into two parts:

- Predict $P(d_{\text{offset}}|c_{sk}, s)$, where d_{offset} describes a deformation of c_{sk} by offsets, and c_{sk} is a skeleton image generated from the content image c.
- Predict $P(y|d, s)$, where d is the deformation of c_{sk} after each offset in d_{offset} are applied.

d_{offset} is represented by $v \times h \times w$ and d is represented by $b \times H \times W$. The difference between the architecture used to predict $P(d_{\text{offset}}|c_{sk}, s)$ and $P(y|d, s)$ is:

- The output of the decoder has 3 channels instead of 1.
- Discriminators have adjusted input channel size to match the output of the decoder.
- The content and ground truth images are different.

The reason the output of the decoder has 3 channels, while 2 would be sufficient to represent the offset vectors, is simply because $3 \times h \times w$ can be represented as a colored image, while $2 \times h \times w$ can not. This makes it easier to troubleshoot, for instance the output of the decoder can be visualized during training. This modification is named Deformation and Texture Separation (DTS).

5. Experiments and Results

In this chapter, the results of each experiment are presented. Section 5.1 describes what experiments are conducted and how they are evaluated. Section 5.2 describes the setup for the experiments. Section 5.3 describes the results of the experiments. The results will be discussed in more detail in Chapter 6.

5.1. Experimental Plan

As mentioned in Section 1.2, the thesis goal is to reduce the amount of unrecognizable characters generated in a font style transfer method. To check whether the goal is achieved, experiments which aim at answering the research questions (described in Section 1.2) are conducted. Additionally, how the method performs compared to the state-of-the-art is tested. To evaluate each experiment, quantitative evaluations of the experiments are performed. For the comparison of the method to the state-of-the-art in Section 6.4, qualitative evaluations are conducted as well.

For the quantitative evaluations, five different evaluation metrics is used; Mean absolute error (MAE), ResNet-50 classification of content, ResNet-50 classification of style, Learned Perceptual Image Patch Similarity (LPIPS) and Fréchet Inception Distance (FID). MAE gives a general indication on the performance on the model. However, MAE is not a good indication to whether the images are easy to classify or pleasing to look at for humans, which the job of the other metrics. Similar to other few-shot font style transfer methods (Li et al., 2021; Park et al., 2021b), two ResNet-50 is trained, one for classifying the content of a glyph image and one for classifying the style of a glyph image. ResNet-50 is a strong classification model proposed by He et al. (2016). The top-1 classification accuracy of the content and the style is reported. Lastly, LPIPS and FID is used to measure the similarity between ground truth and generated images in a way which correlates with human judgement (Zhang et al., 2018a; Heusel et al., 2017).

For the qualitative evaluations, user studies are conducted. The participants are asked which generated image out of each model is the most satisfactory in terms of style matching degree and content recognizability. One generated image is made for each font in the test set. The qualitative evaluations will only be used when comparing the model to the state-of-the-art. This is because the evaluation in this experiment is prioritized over other experiments.

To answer **RQ1**, experiments with different approaches on content feature and style feature extractions are performed.

To answer **RQ2**, experiments with different settings for applying Deep Metric Learning are performed.

To answer **RQ3**, experiments on EPS-Font modified with the proposed modification Deformation and Texture Separation (DTS) are performed.

5.2. Experimental Setup

The dataset proposed by Li et al. (2021) is used to test the proposed method. It contains 847 gray-scale fonts (style inputs) each with approximately 1000 commonly used Chinese characters and 52 Latin letters in the same style. The font style used for content reference is Microsoft Yahei, which is described as “A Simplified Chinese font developed by taking advantage of ClearType technology, and it provides excellent reading experience particularly onscreen” [1]. Intuitively, it being simplified should mean that disentangling the content features from its style features is easier, and it “providing excellent reading experience” should mean that it will accurately represent the characters. However, previous work shows that the choice of font style for content images does not significantly impact the final results (Zhu et al., 2020; Gao et al., 2019). 29 of the fonts are set aside for testing.

The weights in the loss function are: $\lambda_1 = 100.0$, $\lambda_s = 1.0$, $\lambda_c = 1.0$. λ_1, λ_s and λ_c are the same weights as in the pix2pix implementation. The pix2pix implementation is described in Section 2.8. For the experiments featuring Deep Metric Learning, $\lambda_{SC} = 10$ is used. λ_{SC} is set high to make up for the difference in learning rate between these experiments and experiments by Xuan et al. (2020), who proposed L_{SC}. The batch size for L_1, $L_{content}$ and L_{style} is 64, while the batch size for L_{SC} is 512. Preferably, one would use a higher batch size than 64 if possible (Brock et al., 2018). The batch size for L_{SC} can be set high because of the way it is implemented, which is described in Section 4.5. The same configuration as used in Xuan et al. (2020)’s experiments is used for the experiments, i.e., with $m = 0.2$ and $\alpha = 1$. The model is trained for 20 epochs with the amount of styles $K = 6$ and learning rate $lr = 0.0002$, which is the same parameters as Li et al. (2021)’s experiments with Few-shot Font Style Transfer between Different Languages (FTransGAN). English glyphs are used as style reference images, while Chinese glyphs are used for content images and ground truth.

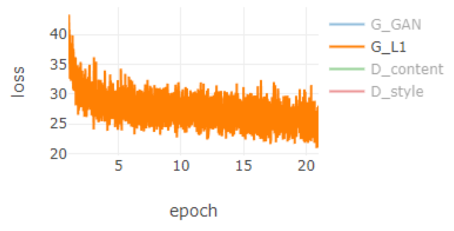
5.3. Experimental Results

In this section, the training details of EPS-Font are presented. Afterwards, a comparison of the effectiveness of different approaches for the encoders, the effectiveness of Deep Metric Learning and the result of using Deformation and Texture Separation (DTS) are given.

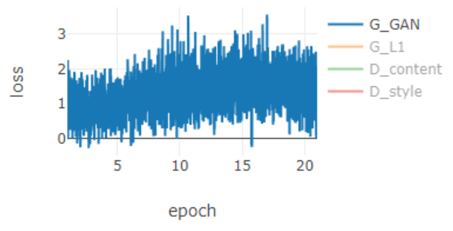
5.3.1. Training Details

Figure 5.1 shows the plots when running the model, and a generated example of every 5th epoch is shown in Figure 5.2.

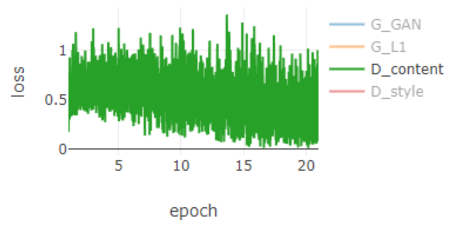
[1] https://docs.microsoft.com/en-us/typography/font-list/microsoft-yahei



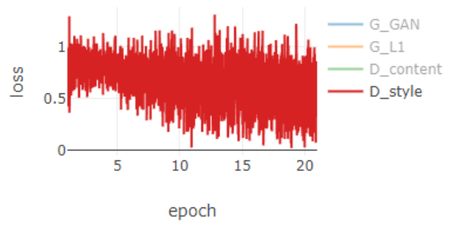
(a) L_{L1}



(b) $L_{contentG} + L_{styleG}$



(c) $L_{contentD}$



(d) L_{styleD}

Figure 5.1.: Losses during training of EPS-Font.

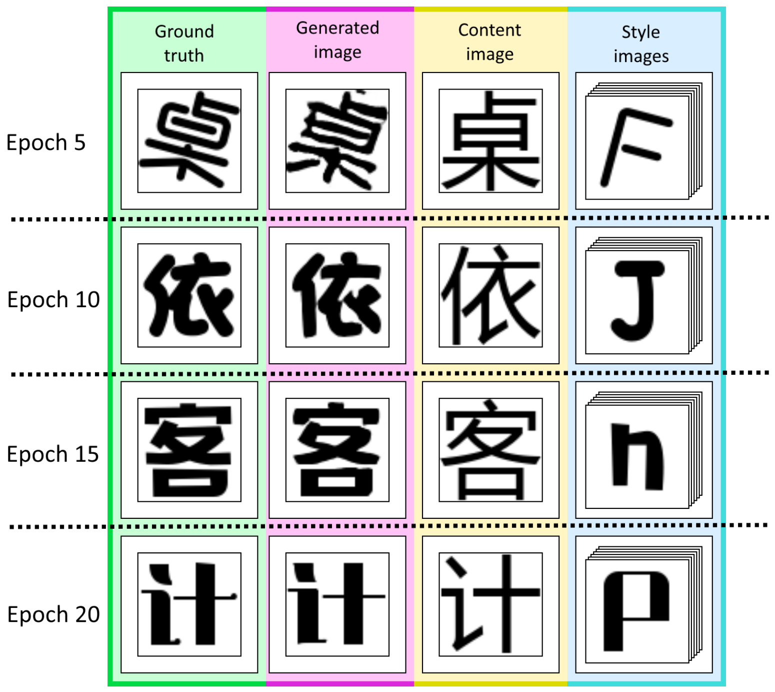


Figure 5.2.: Example of generated images during training. Columns show ground truth, generated image, content image and style images, respectively, from left to right. Rows show every fifth epoch.

5.3.2. Ablation Studies

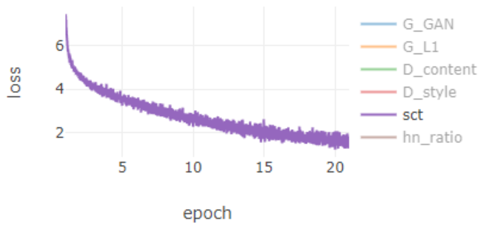
Encoders These experiments relate to **RQ1**. The goal is to test how the different approaches of extracting content feature and style features affect few-shot font style methods in general.

As mentioned in Section 4.2, the model has deliberately a deeper style encoder than context encoder, which differs from many font generation models (Zhang et al., 2018b; Zhu et al., 2020; Gao et al., 2019). Additionally, the style encoder uses a convolutional layer for downsampling the style features to a spatial dimension of $\{1 \times 1\}$, then upsamples the style features to match the spatial dimension of the content features. This also differs from many font generation models (Zhang et al., 2018b; Zhu et al., 2020; Gao et al., 2019; Park et al., 2021b). To test the effect of each of these choices, three respective experiments are conducted. In Experiment 1, the same number of convolutional layers on the content encoder as the style encoder is used. In Experiment 2, the last convolutional layer and the upsample layer are removed from the style encoder. Experiment 3 will test both of these changes at once, which is a typical font style transfer architecture (Zhang et al., 2018b; Zhu et al., 2020; Gao et al., 2019). A detailed overview of the architecture for Experiment 1, Experiment 2 and Experiment 3 can be found in Appendix B. Table 5.1 demonstrates the results of EPS-Font and the results of the experiments.

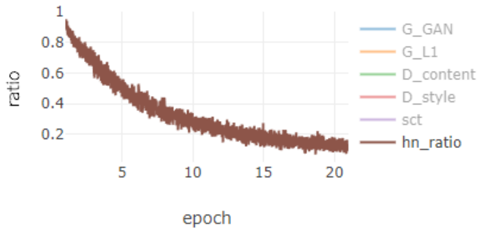
Table 5.1.: Experiments testing the effect of different encoder architectures

Model	Acc(C) %	Acc(S) %	MAE	LPIPS	FID
EPS-model	**98.3**	**10.7**	**0.182**	**0.128**	**17.40**
Exp. 1: Modified content encoder	95.0	10.5	0.302	0.130	20.62
Exp. 2: Modified style encoder	97.9	9.8	0.194	0.142	21.32
Exp. 3: Modified both encoders	94.2	9.4	0.327	0.146	24.45

Deep Metric Learning These experiments relate to **RQ2**. Figure 5.1 shows the L_{SC} and the ratio of hard negatives when running EPS-Font with $\lambda_{SC} = 10, m = 0.2, \alpha = 1$. The rest of the plots are shown in Appendix C. A common technique to analyze the effect of a Deep Metric Learning is t-Distributed Stochastic Neighbor Embedding (t-SNE). T-SNE visualizes high dimensional data as a two or three-dimensional map. Figure 5.4 shows the t-SNE for EPS-Font with $\lambda_{SC} = 0$ and EPS-Font with $\lambda_{SC} = 10$, $m = 0.2$, $\alpha = 1$. Table 5.2 shows runs with different settings for λ_{SC}, m and α.

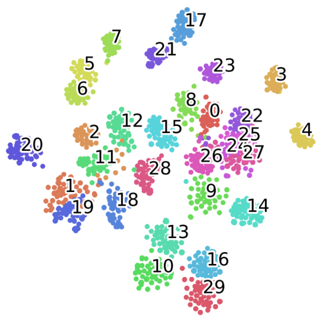


(a) L_{SC}.

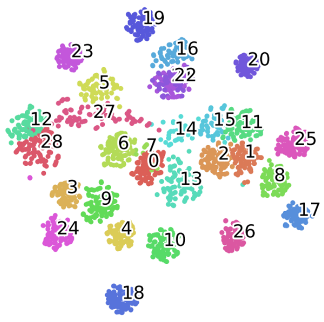


(b) Ratio of hard negatives

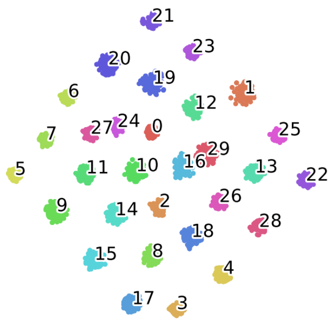
Figure 5.3.: L_{SC} and ratio of hard negatives during training of EPS-Font with $\lambda_{SC} = 10$, $m = 0.2$, $\alpha = 1$.



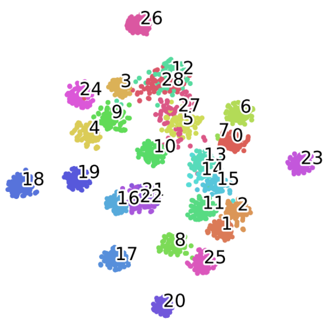
(a) Result with $\lambda_{SC} = 0$. 30 fonts (train)



(b) Result with $\lambda_{SC} = 0$. 29 fonts (test)



(c) Result with $\lambda_{SC} = 10$, $m = 0.2$, $\alpha = 1$. 30 fonts (train)



(d) Result with $\lambda_{SC} = 10$, $m = 0.2$, $\alpha = 1$. 29 fonts (test)

Figure 5.4.: T-SNE of style features of EPS-Font.

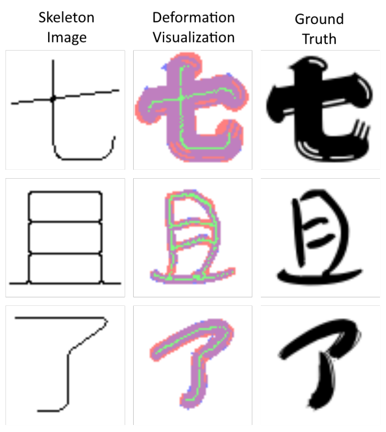
Table 5.2.: Experiments testing the effect of deep metric learning

Model	Settings	Acc(C) %	Acc(S) %	MAE	LPIPS	FID
EPS-Font	$\lambda_{SC} = 0$	**98.3**	**10.7**	**0.182**	**0.128**	**17.40**
EPS-Font	$\lambda_{SC} = 10$, $m = 0.2$, $\alpha = 1$	97.0	10.1	0.219	0.132	18.40
EPS-Font	$\lambda_{SC} = 10$, $m = 0.2$, $\alpha = 0.5$	97.2	10.4	0.210	0.130	18.18
EPS-Font	$\lambda_{SC} = 10$, $m = 0$, $\alpha = 1$	95.1	10.6	0.254	0.144	20.09
EPS-Font	$\lambda_{SC} = 5$, $m = 0.2$, $\alpha = 1$	97.7	10.4	0.189	0.129	18.14
EPS-Font	$\lambda_{SC} = 5$, $m = 0$, $\alpha = 0.5$	96.3	**10.7**	0.217	0.136	18.79

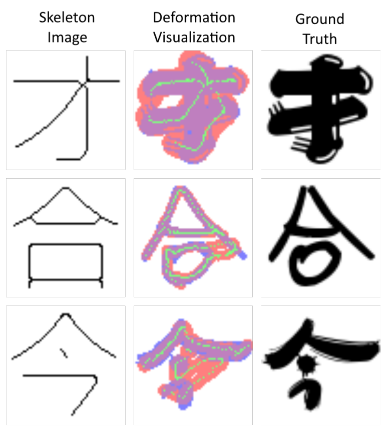
Deformation and Texture Separation These experiments relate to **RQ3**. This modification is described in Section 4.6. There are two important results:

- The results of $\mathcal{T}$
- The results of EPS-Font with DTS

For $\mathcal{T}$, the configuration used is $\alpha_{\text{dist}} = 8$, $\alpha_{\text{step}} = 5$, $\beta_{\text{pos}} = \beta_{\text{angle}} = 0.5$ and $n_{\text{steps}} = 30$. To find these parameter values, grid random search was used on a subset of the dataset, with $\lambda = 1$. Using $\mathcal{T}$ on all glyphs in the dataset took ~ 7 days. Figure 5.5 shows some examples of deformations using $\mathcal{T}$.



(a) Good deformations



(b) Bad deformations

Figure 5.5.: Deformation examples generated by $\mathcal{T}$. The green pixels show d, the deformation of c_{sk}. The purple pixels show the overlap between the estimated stroke used in $\mathcal{T}$ and the ground truth. The red and blue pixels show the estimated stroke used in $\mathcal{T}$ and the ground truth respectively with no overlap.

EPS-Font was used with the same configurations as mentioned in Section 5.1, only the architecture was modified for $P(d_{\text{offset}}|c_{sk}, s)$ and $P(x|d, s)$. Table 5.3 demonstrates the results of this experiment.

Table 5.3.: Experiment testing the EPS-Font with Deformation and Texture Separation

Model	Acc(C) %	Acc(S) %	MAE	LPIPS	FID
EPS-Font	**98.3**	**10.7**	**0.182**	**0.128**	**17.40**
EPS-Font /w DTS	72.5	5.2	0.361	0.193	38.51

6. Evaluation and Discussion

In this chapter, a discussion of the experimental results in Chapter 5 are presented. Section 6.1 discusses the experimental results of the experiments of different encoder architectures. Section 6.2 discusses the experimental results of applying Deep Metric Learning to few-shot font style transfer with Extraction of Partial Style (EPS-Font). Section 6.3 discusses the experimental results of the modification Deformation and Texture Separation (DTS). Section 6.4 compares EPS-Font to the state-of-the-art.

6.1. Encoders

This discussion relates to **RQ1**. Using the same architecture for the content and style encoder is tempting, as it is the easiest approach in terms of implementation and documentation. This approach is used by several font style transfer methods (Zhang et al., 2018b; Zhu et al., 2020; Gao et al., 2019). However, this approach also has the worst performance out of all experiments in Section 5.3.2. The results indicate that modifying the encoders with the suggested changes in Section 4 improves the overall performance of the generator, as each change clearly improves the overall evaluations in the experiments. The first suggested change is to use less convolutional layers on the content encoder than the style encoder. The second suggested change is to use a convolutional layer for downsampling the style features to a spatial dimension of $\{1 \times 1\}$, then upsample it to match the spatial dimension of the content features. The reasoning behind these suggestions is mentioned in Section 4.

6.2. Deep Metric Learning

This discussion relates to **RQ2**. The experiments performed to test the effects of Deep Metric Learning (DML) suggest that DML had negative effects when applied to EPS-Font. DML is described in Section 2.9. The model with the best overall results is EPS-Font with $\lambda_{SC} = 0$, where DML was not used. The results in Table 5.2 and t-SNE in Figure 5.4 display indications as to why DML did not improve the results of EPS-Font.

Firstly, the ResNet-50 classifier has a significantly low score on predicting the style generally in Table 5.2. This could mean that the models tested are not able to transfer the styles properly. However, it is less likely, as the models show comparable score to Multiple Localized Experts Few-shot Font Generation Network (MX-Font) in Park et al. (2021b)'s paper proposing MX-Font on all other evaluations, specifically, on their experiments on font style transfer between Chinese glyphs. Still, MX-Font scored much higher on style classification in their experiments. The reason for this is likely that the dataset used in the experiments presented in Chapter 5 has more styles which are similar to another, which makes the task of classifying the styles harder as well. One pretty evident indication of this is that the dataset used in these experiments has almost double the amounts of styles (847 compared to 495). A possible explanation for this is that the more similar styles there are in a dataset, the worse DML performs for font style transfer. This is because DML tries to force a margin between similar styles in the embedded space. This is useful in a task like style classification, where DML is typically used (Schroff et al., 2015; Hermans et al., 2017; Xuan et al., 2020). A misclassification means that the output is incorrect, therefore, for style classification it is likely beneficial that DML encourages the model to separate between similar styles in the embedded space. However, font style transfer is not the same as style classification. For font style transfer, even if the style encoder is unable to differentiate between two similar styles, it could still achieve good results as long as the generated image ends up resembling the target image. For this reason, using DML to encourage the model to separate between similar styles in the embedded space is not necessarily as beneficial for font style transfer. Instead, this can hurt the model's capabilities at generalizing styles and generating glyphs, as part of the model's focus is instead directed at learning to separate between similar styles.

Secondly, the t-Distributed Stochastic Neighbor Embedding (t-SNE) of EPS-Font with $\lambda_{SC} = 0$ shows a rather ideal t-SNE. The t-SNEs are shown in Figure 5.4a and Figure 5.4b. The reason for this is that the majority of style features can be easily separated from each other, both on the examples of the training set and on the test set. Additionally, the style features are spread as a manifold, which indicates that the model is able to generalize (Schroff et al., 2015). The aim of DML is to optimize the embeddings (Schroff et al., 2015); however, when the embeddings are already rather ideal without DML, the benefits of using DML is lessened.

As mentioned in Section 3.5, Aoki et al. (2021) showed that DML improved the overall results in their experiments. However, their experiments differ from the experiments conducted here. Their dataset consists of 368 fonts, which is less than half the fonts (847) used in the experiments conducted in Chapter 5. This reduces the likelihood of similar fonts. Additionally, they fine-tuned when testing their implementation of DML on AGIS-Net(Gao et al., 2019). In Figure 5.4c, it is noticeable that DML made it very easy to clearly separate the features of the training set. However, the same can not be said in Figure 5.4d, the test set. This suggests that a model with DML generalizes worse for unseen fonts. As Aoki et al. fine-tune the model, they do not test on unseen fonts.

6.3. Deformation and Texture Separation

This discussion relates to **RQ3**. The results of using EPS-Font with Deformation and Texture Separation (DTS) is pretty evidently inferior to using EPS-Font without DTS. This is likely because many of the deformations given by the mapping $\mathcal{T}$ were inaccurate, as shown in Figure 5.5. The inaccuracy in deformations mapped from T both affected the model's ability to generalize the deformation at $P(d_{\text{offset}}|c_{sk}, s)$, and the overall performance of font generation at $P(y|d, s)$.

However, one interesting use case of Deformation and Texture Separation (DTS) is to use c_{sk} instead of d as input for $P(x|d, s)$. Figure 6.1 shows an example of this use case. This way of generating images removes the deformation of the style. Definitely, one way to use this could be for real-time applications to improve the readability of fonts that have strong deformations. However, a perhaps more practical usage of this is to use this as a tool for font generation. We denote $s_{d=0}$ as a style image with no deformation generated through this method.

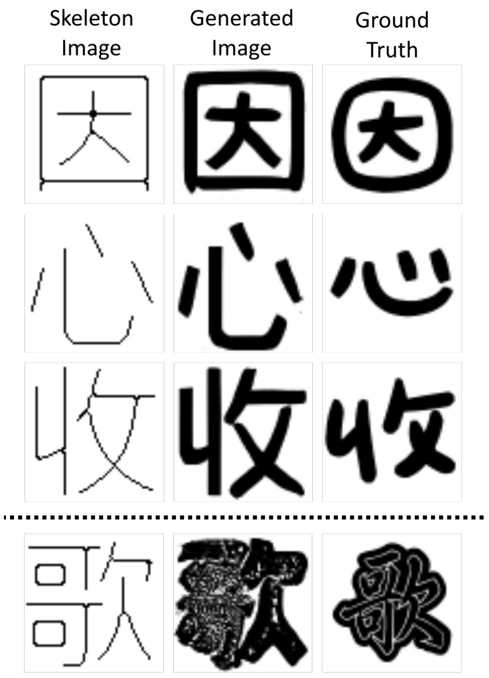


Figure 6.1.: Example of removing deformation from fonts using approach 1. Top three rows show successful cases. The bottom row shows a failure case.

One way to make use of $s_{d=0}$, is for splitting the task of predicting the deformation and texture into two separate models. Learning only the texture when there is no deformation is much easier. Gao et al. (2019) wrote in their paper which proposes AGIS-Net, a state-of-the-art font generation model: ”The task of shape style transfer is much tougher than texture style transfer. [...] Actually, not only the proposed model but also all other existing approaches can not satisfactorily transfer the styles of some fonts, in which most characters have their own unique shape style or/and local details.”. The great results of state-of-the-art style transfer methods (Chen and Schmidt, 2016; Johnson et al., 2016; Huang and Belongie, 2017) also indicate that learning to predict the texture is easier when there is no deformation. Style transfer is similar to font style transfer, with the exception of how style is defined in each of the tasks. In style transfer, style is defined as a set of colors and texture, while in font style transfer, stroke, shape, decorations, etc. are also regarded as style. One way to make use of $s_{d=0}$ is by using $s_{d=0}$ as ground truth for learning the texture, then mapping $s_{d=0} \to s$ by another model. Learning highly artistic styles are very hard, even for state-of-the-art few-shot font style transfer methods (Gao et al., 2019; Li et al., 2021), but with the help of $s_{d=0}$ it could be more approachable.

However, as shown in the bottom row of Figure 6.1, the model is not able to consistently successfully remove the deformation. This is likely because the deformation made by $\mathcal{T}$ is too inaccurate, as mentioned previously.

6.4. Comparison with the State-of-the-Art

The options of competitors are very limited as the only other font style transfer approach that transfers between different languages is FTransGAN. Few other font generation methods can be modified for this task. Li et al. (2021) chose EMD (Zhang et al., 2018b) and DFS (Zhu et al., 2020) as their competitors, after excluding:

- Models that are designed for compositional scripts.
- Models that can not handle large font libraries.
- Models designed for unsupervised generation.

As FTransGAN is able to clearly demonstrate that it outperforms EMD and DFS, only FTransGAN is compared to EPS-Font. Table 6.1 shows the result of this comparison and Figure 6.2 shows examples of generated images from FTransGAN and EPS-Font. The results of FTransGAN are not the results published in Li et al.'s paper, but the results with the experimental setups described in Section 5.2. Notably, the batch size in these experiments was lower, which is not preferred (Brock et al., 2018).

A total of 232 responses from 8 people were collected by user studies. However, none of the participants were proficient in Chinese.

Table 6.1.: Comparison of EPS-Font and FTransGAN (Li et al., 2021)

Model	User %	Acc(C) %	Acc(S) %	MAE	LPIPS	FID
EPS-Font	**60.8**	**98.3**	**10.7**	**0.182**	0.128	**17.40**
FTransGAN	39.2	98.0	10.2	0.188	**0.126**	19.07

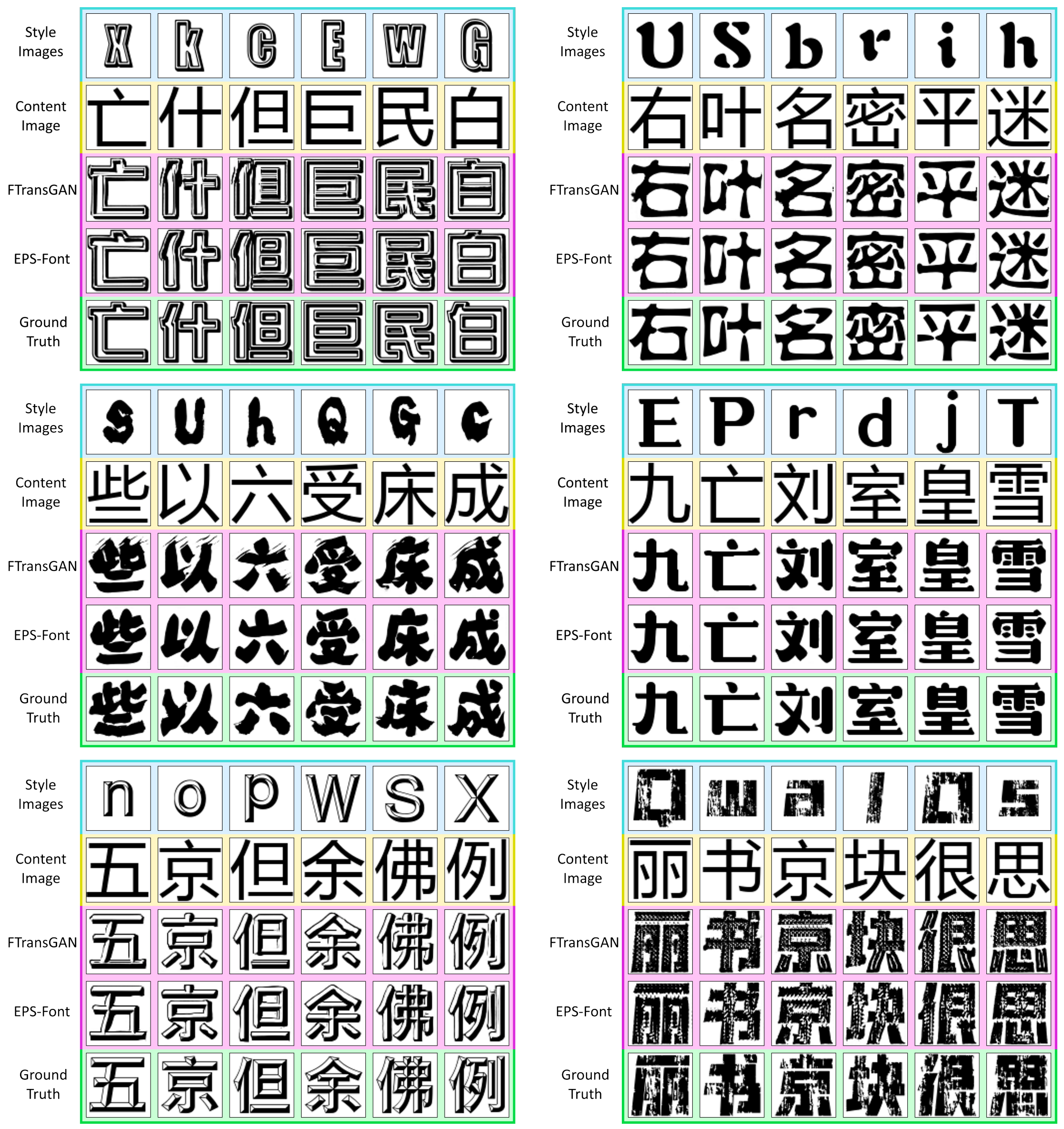


Figure 6.2.: Generated examples from FTransGAN and EPS-Font. Both models are able to extract the style well. However, the generated images from EPS-Font are often sharper than the generated images from FTransGAN.

7. Conclusion and Future Work

This chapter concludes the thesis with the contributions to the state-of-the-art and future work. Section 7.1 describes each of the contributions to the state-of-the-art by their related research question. Section 7.2 presents the overview of future work.

7.1. Contributions

RQ1: Using the architecture of the encoders in the outlined 'Few-shot font style transfer with Extraction of Partial Style' (EPS-Font) approach is proposed as a way to extract content and style features in few-shot font style transfer. The results of experiments conducted shows that this architecture greatly increases the overall performance compared to using a typical architecture, which is using the same architecture for the content encoder and style encoder. Experimental results show that EPS-Font outperforms state-of-the-art methods in both quantitative and qualitative evaluations.

RQ2: The implementation of Deep Metric Learning was not beneficial to improve the results of EPS-Font. A possible explanation for this is that the more similar fonts in a dataset, the more of the focus of EPS-Font is shifted to separating these fonts in the embedded space, rather than learning to generalize the styles.

RQ3: The proposed modification, Deformation and Texture Separation (DTS) showed to have bad results when used for few-shot font style transfer. However, this modification enables a way of removing deformation from a style image, which has not been accomplished before. Style images with no deformation can be beneficial for training font style transfer methods.

7.2. Future Work

A challenge within the font style transfer field is to predict highly artistic font styles. This is prominent in the models tested in this thesis, as shown in Figure 7.1. Prediction of highly artistic font styles is an interesting task to look further into.

Testing out different approaches like AdaIN and U-Net, which both show great results at preserving high frequency details (Huang and Belongie, 2017; Ronneberger et al., 2015), can be useful to experiment on the architecture of EPS-Font.

The mapping $\mathcal{T}$ for the modification, Deformation and Texture Separation (DTS), was likely too inaccurate. Finding a better way to do $\mathcal{T}$ would be interesting to improve DTS

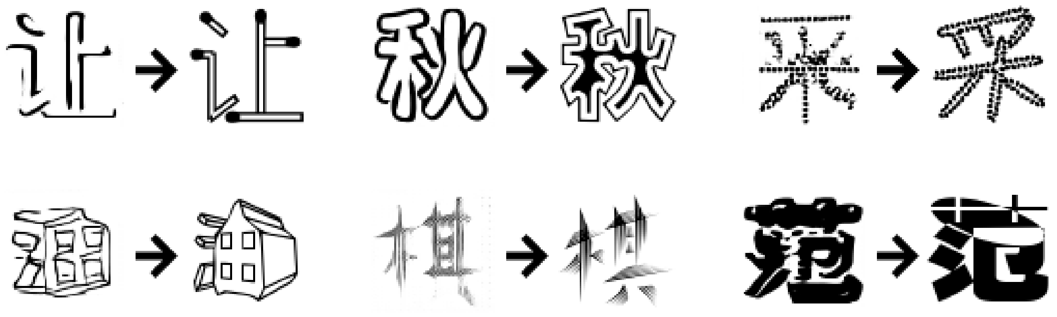


Figure 7.1.: Examples of failure cases with highly artistic font styles. The left image in each pair is the generated image, the right image in each pair is the ground truth.

in overall. For instance, an approach which makes use of neural networks can make $\mathcal{T}$ more flexible overall. Additionally, the gradients will be maintained, which can make it possible for the architecture used to predict $P(d_{\text{offset}}|c_{sk}, s)$ to be trained jointly with the architecture used to predict $P(y|d, s)$. These architectures are described in Chapter 4.6.

Comparing this model against more state-of-the-art models would be highly beneficial to weight the advantages and disadvantages of state-of-the-art approaches, even if the font style transfer experiments are not performed cross language. For instance, testing EPS-Font against Few-shot Font Generation with Multiple Localized Experts (MX-Font) for font style transfer of Chinese letters.

Bibliography

Haruka Aoki, Koki Tsubota, Hikaru Ikuta, and Kiyoharu Aizawa. Few-shot font generation with deep metric learning. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 8539–8546, 2021. doi:10.1109/ICPR48806.2021.9412254.

Samaneh Azadi, Matthew Fisher, Vladimir G. Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell. Multi-content GAN for few-shot font style transfer. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7564–7573, 2018.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis, 2018. URL `https://arxiv.org/abs/1809.11096`.

A. Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019. ISBN 9781999579517. URL `https://books.google.no/books?id=0jbxwQEACAAJ`.

Neill D. F. Campbell and Jan Kautz. Learning a manifold of fonts. *ACM Transactions on Graphics (TOG)*, 33:1–11, 2014.

Prashanth Chandran, Gaspard Zoss, Paulo Gotardo, Markus Gross, and Derek Bradley. Adaptive convolutions for structure-aware style transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7972–7981, Jun 2021.

Tian Qi Chen and Mark Schmidt. Fast patch-based style transfer of arbitrary style, 2016. URL `https://arxiv.org/abs/1612.04337`.

Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 764–773, 2017.

A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition, 2015. ISBN 3662448734.

Fartash Faghri, David J. Fleet, Jamie Ryan Kiros, and Sanja Fidler. VSE++: Improving visual-semantic embeddings with hard negatives. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2018.

Yue Gao, Yuan Guo, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. Artistic glyph image synthesis via one-stage few-shot learning. *ACM Transactions on Graphics*, 38(6):1–12, Nov 2019. ISSN 1557-7368. doi:10.1145/3355089.3356574. URL `http://dx.doi.org/10.1145/3355089.3356574`.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016. doi:10.1109/CVPR.2016.265.

Claudio Gentile and Manfred K. K Warmuth. Linear hinge loss and average margin. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11. MIT Press, 1998. URL `https://proceedings.neurips.cc/paper/1998/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf`.

Manfred Gilli and Enrico Schumann. A note on ‘good starting values’ in numerical optimisation. *SSRN Electronic Journal*, Jun 2010. doi:10.2139/ssrn.1620083.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with Hilbert-Schmidt norms. In Sanjay Jain, Hans Ulrich Simon, and Etsuji Tomita, editors, *Algorithmic Learning Theory*, pages 63–77, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31696-1.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *CoRR*, abs/1703.07737, 2017. URL `http://arxiv.org/abs/1703.07737`.

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. 2017. doi:10.48550/ARXIV.1706.08500.

Xun Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1510–1519, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2017.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016. URL `http://arxiv.org/abs/1603.08155`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi:https://doi.org/10.1002/nav.3800020109. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109`.

Chenhao Li, Yuta Taniguchi, Min Lu, and Shin'ichi Konomi. Few-shot font style transfer between different languages. *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 433–442, 2021.

Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2813–2821, 2017.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014. URL `http://arxiv.org/abs/1411.1784`.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018. URL `http://arxiv.org/abs/1802.05957`.

Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier GANs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651. PMLR, Aug 2017. URL `https://proceedings.mlr.press/v70/odena17a.html`.

Song Park, Sanghyuk Chun, Junbum Cha, Bado Lee, and Hyunjung Shim. Few-shot font generation with localized style representations and factorization. In *Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)*, 2021a.

Song Park, Sanghyuk Chun, Junbum Cha, Bado Lee, and Hyunjung Shim. Multiple heads are better than one: Few-shot font generation with multiple localized experts. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 13880–13889, 2021b.

Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2536–2544, 2016.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2018. URL `https://openreview.net/forum?id=SkBYYyZRZ`.

Rajeev Ranjan, Carlos Domingo Castillo, and Rama Chellappa. L2-constrained softmax loss for discriminative face verification. *CoRR*, abs/1703.09507, 2017. URL `http://arxiv.org/abs/1703.09507`.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL `http://arxiv.org/abs/1505.04597`.

Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural computation*, 16:1063–76, Jun 2004. doi:10.1162/089976604773135104.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun 2015. doi:10.1109/cvpr.2015.7298682.

Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4004–4012, 2016.

Rapee Suveeranont and Takeo Igarashi. Example-based automatic font generation. In Robyn Taylor, Pierre Boulanger, Antonio Krüger, and Patrick Olivier, editors, *Smart Graphics*, pages 127–138, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-13544-6.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. *CoRR*, abs/1611.02200, 2016. URL `http://arxiv.org/abs/1611.02200`.

Paul Upchurch, Noah Snavely, and Kavita Bala. From A to Z: supervised transfer of style and content using deep neural network generators. *CoRR*, abs/1603.02003, 2016. URL `http://arxiv.org/abs/1603.02003`.

Yankun Xi, Guoli Yan, Jing Hua, and Zichun Zhong. JointFontGAN: Joint geometry-content GAN for font generation via few-shot learning. *Proceedings of the 28th ACM International Conference on Multimedia*, pages 4309–4317, 2020.

Yangchen Xie, Xinyuan Chen, Li Sun, and Yue Lu. DG-Font: Deformable generative networks for unsupervised font generation. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5126–5136, 2021.

Hong Xuan, Abby Stylianou, Xiaotong Liu, and Robert Pless. Hard negative examples are hard, but useful. In *ECCV 2020*, pages 126–142, 2020.

Shuai Yang, Jiaying Liu, Zhouhui Lian, and Zongming Guo. Awesome typography: Statistics-based text effects transfer. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2886–2895, 2017.

Tian Yuchen. zi2zi: Master chinese calligraphy with conditional adversarial networks, 2017. URL https://github.com/kaonashi-tyc/zi2zi.

Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7354–7363. PMLR, Jun 2019. URL https://proceedings.mlr.press/v97/zhang19d.html.

Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. In *ECCV 2016*, pages 649–666, 2016.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018a.

Yexun Zhang, Wenbin Cai, and Ya Zhang. Separating style and content for generalized style transfer. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8447–8455, 2018b.

Anna Zhu, Xiongbo Lu, Xiang Bai, Seiichi Uchida, Brian Kenji Iwana, and Shengwu Xiong. Few-shot text style transfer via deep feature similarity. *IEEE Transactions on Image Processing*, 29:6932–6946, 2020. doi:10.1109/TIP.2020.2995062.

A. Algorithms

Psuedocode for estimating the stroke in Deformation and Texture Separation.

Algorithm 1 Stroke Estimation

```
procedure ESTIMATE-STROKE($c_{sk}$, $y$)    ▷ $c_{sk}$ is the skeleton image, $y$ is the target image
    $Stroke \leftarrow 4$
    do
        $Stroke_{old} \leftarrow Stroke$
        $\hat{y} \leftarrow$ ADD-STROKE-TO-SKELETON($c_{sk}$)
        $\nabla \leftarrow$ COUNT-BLACK-PIXELS($y$) / COUNT-BLACK-PIXELS($\hat{y}$)
        $Stroke \leftarrow Stroke * \nabla^{\frac{1}{2}}$
    while $|Stroke - Stroke_{old}| > 0.25$    ▷ Change in stroke must be less than 0.25
    return $Stroke$
end procedure
procedure COUNT-BLACK-PIXELS($p$)
    local variables: $n$, images are $n \times n$
    return $n^2 - \sum_{y=1}^{n} \sum_{x=1}^{n} p_{x,y}$
end procedure
```

B. Architectures

Detailed overview of the architectures. Sections B.1-B.3 show the architecture of EPS-Font. Sections B.4-B.6 show the architectures of the encoders used in Section 5.3.2.

B.1. Encoders

	Operation	Kernel	Stride	Padding	Feature	Normalization	Activation
Content encoder	Convolution	7	1	3	64	BN	ReLU
	Convolution	3	2	1	128	BN	ReLU
	Convolution	3	2	1	256	BN	ReLU
Style encoder	Convolution	7	1	3	64	BN	ReLU
	Convolution	3	2	1	128	BN	ReLU
	Convolution	3	2	1	256	BN	ReLU
	Convolution	3	2	1	512	BN	ReLU
	Convolution	3	2	1	1024	BN	ReLU
	Convolution	4	1	0	256	BN	ReLU
	Convolution	1	1	0	256	BN	ReLU
	Upsample	4	-	-	256	-	-

B.2. Decoder

Operation	Kernel	Stride	Padding	Feature	Normalization	Activation
ResBlock	3	1	1	512	BN	-
ResBlock	3	1	1	512	BN	-
ResBlock	3	1	1	512	BN	-
ResBlock	3	1	1	512	BN	-
ResBlock	3	1	1	512	BN	-
ResBlock	3	1	1	512	BN	-
ConvTranspose	3	2	1	256	BN	ReLU
ConvTranspose	3	2	1	128	BN	ReLU
Convolution	7	1	3	1	-	Tanh

B.3. Discriminator

Operation	Kernel	Stride	Padding	Feature	Normalization	Activation
Convolution	4	2	1	64	-	LeakyReLU(0.2)
Convolution	4	2	1	128	BN	LeakyReLU(0.2)
Convolution	4	2	1	256	BN	LeakyReLU(0.2)
Convolution	4	1	1	512	BN	LeakyReLU(0.2)
Convolution	4	1	1	1	-	-

B.4. Experiment 1

	Operation	Kernel	Stride	Padding	Feature	Normalization	Activation
Content encoder	Convolution	7	1	3	64	BN	ReLU
	Convolution	3	1	1	128	BN	ReLU
	Convolution	3	2	1	256	BN	ReLU
	Convolution	3	1	1	512	BN	ReLU
	Convolution	3	2	1	1024	BN	ReLU
	Convolution	3	1	1	256	BN	ReLU
Style encoder	Convolution	7	1	3	64	BN	ReLU
	Convolution	3	2	1	128	BN	ReLU
	Convolution	3	2	1	256	BN	ReLU
	Convolution	3	2	1	512	BN	ReLU
	Convolution	3	2	1	1024	BN	ReLU
	Convolution	4	1	0	256	BN	ReLU
	Convolution	1	1	0	256	BN	ReLU
	Upsample	4	-	-	256	-	-

B.5. Experiment 2

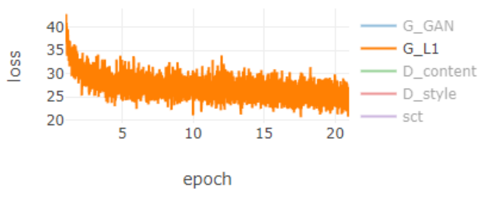
	Operation	Kernel	Stride	Padding	Feature	Normalization	Activation
Content encoder	Convolution	7	1	3	64	BN	ReLU
	Convolution	3	2	1	128	BN	ReLU
	Convolution	3	2	1	256	BN	ReLU
Style encoder	Convolution	7	1	3	64	BN	ReLU
	Convolution	3	1	1	128	BN	ReLU
	Convolution	3	2	1	256	BN	ReLU
	Convolution	3	1	1	512	BN	ReLU
	Convolution	3	2	1	1024	BN	ReLU
	Convolution	3	1	1	256	BN	ReLU

B.6. Experiment 3

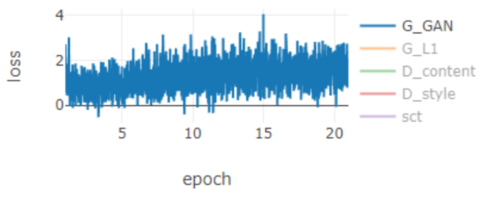
	Operation	Kernel	Stride	Padding	Feature	Normalization	Activation
Content encoder	Convolution	7	1	3	64	BN	ReLU
	Convolution	3	1	1	128	BN	ReLU
	Convolution	3	2	1	256	BN	ReLU
	Convolution	3	1	1	512	BN	ReLU
	Convolution	3	2	1	1024	BN	ReLU
	Convolution	3	1	1	256	BN	ReLU
Style encoder	Convolution	7	1	3	64	BN	ReLU
	Convolution	3	1	1	128	BN	ReLU
	Convolution	3	2	1	256	BN	ReLU
	Convolution	3	1	1	512	BN	ReLU
	Convolution	3	2	1	1024	BN	ReLU
	Convolution	3	1	1	256	BN	ReLU

C. Losses

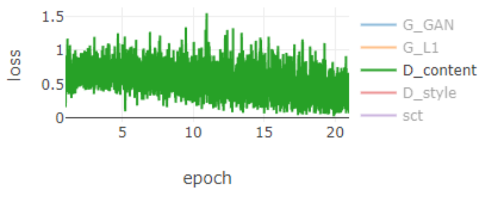
Losses during training of EPS-Font with Deep Metric Learning. The settings used was $\lambda_{SC} = 10$, $m = 0.2$, $\alpha = 1$.



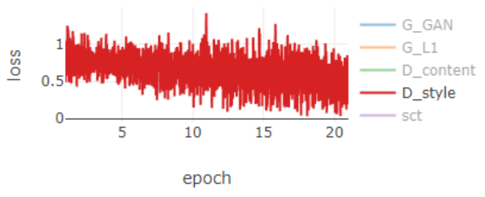
(a) L_{L1}.



(b) $L_{contentG} + L_{styleG}$..



(c) $L_{contentD}$.



(d) L_{styleD}.

Figure C.1.: Losses during training of EPS-Font with $\lambda_{SC} = 10$, $m = 0.2$, $\alpha = 1$.