Scalable Font Reconstruction with Dual Latent Manifolds

Nikita Srivatsan
Language Technologies Institute
Carnegie Mellon University
nsrivats@cmu.edu

Si Wu
Khoury College of Computer Science
Northeastern University
siwu@ccs.neu.edu

Jonathan T. Barron
Google Research
barron@google.com

Taylor Berg-Kirkpatrick
Computer Science and Engineering
University of California, San Diego
tberg@eng.ucsd.edu

Abstract

We propose a deep generative model that performs typography analysis and font reconstruction by learning disentangled manifolds of *both* font style and character shape. Our approach enables us to massively scale up the number of character types we can effectively model compared to previous methods. Specifically, we infer separate latent variables representing character and font via a pair of inference networks which take as input sets of glyphs that either all share a character type, or belong to the same font. This design allows our model to generalize to characters that were not observed during training time, an important task in light of the relative sparsity of most fonts. We also put forward a new loss, adapted from prior work that measures likelihood using an adaptive distribution in a projected space, resulting in more natural images without requiring a discriminator. We evaluate on the task of font reconstruction over various datasets representing character types of many languages, and compare favorably to modern style transfer systems according to both automatic and manually-evaluated metrics.

1 Introduction

The majority of written natural language comes to us in the form of glyphs, visual representations of characters generally rendered in a font with a contextually appropriate style. In order to be legible a glyph must be recognizable as its corresponding character from its underlying shape, but it must also be stylistically consistent with the other glyphs in that font. While this labor intensive process is typically performed manually by human artists,

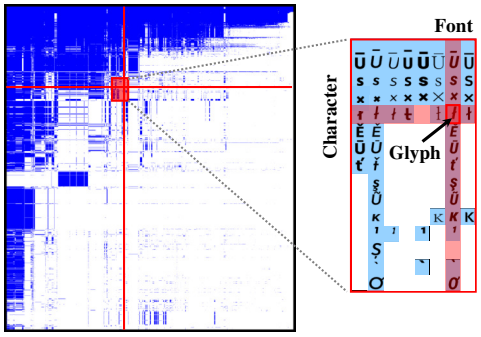


Figure 1: Supported glyphs in Google Fonts organized by character type and font. A blue pixel indicates that column's font includes that row's character. Our proposed model allows font reconstruction over this large, sparse character set.

the number of character types that a font may be expected to support is extremely large, with `Unicode 13.0.0` including as many as 143,859 character types (Unicode). As a result, graphic designers often create glyphs only for a subset of these characters, which tends to be determined by their own cultural context. This can create an accessibility gap for users seeking to create or read digital content in languages with less widespread orthographies, due to the relative lack of available options. Figure 1 shows that within the Google Fonts library (Google) there is a long tail of fonts with a large proportion of missing glyphs.

Font reconstruction is a task that attempts to solve this problem. The goal is for a model, given a small set of example glyphs from an incomplete font, to generate glyphs for the remaining characters in a consistent style. Prior work has approached this in various ways, albeit with some limitations. While some approaches use a variational framework (Srivat-

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 3060–3072
November 7–11, 2021.

san et al., 2019; Lopes et al., 2019), generally their models only treat the font style as a latent variable, and not the character shape. Such methods can therefore only handle a small, fixed, and an a priori known set of character types. Other work such as that by by Zhang et al. (2018); Gao et al. (2019) use discriminative models which dynamically compute both embeddings, allowing them to generalize to unseen characters. However these networks typically require a pre-specified number of observations as input, and their lack of a probabilistic prior can lead to learning a brittle manifold on datasets with a large number of infrequently observed characters.

By contrast, our method learns two smooth manifolds over character shape and font style in order to better share parameters across structurally similar characters, letting it scale to a larger set and more effectively generalize to characters never seen during training. Our model treats font reconstruction as a matrix factorization problem, where we view our corpus as a matrix with rows corresponding to character type, and columns corresponding to fonts. Each row and column is assigned a latent variable that determines its structure or style respectively. A decoder network consisting of transposed convolutional layers parameterizes the model's distribution on each cell in that matrix, *i.e.* an image of a glyph, conditioned on the corresponding row and column embeddings. This approach can be thought of as a generalization of Srivatsan et al. (2019), who used a similar factorization framework, but with only one manifold over font style.

In addition to model structure, the loss function is also important in font reconstruction as pixel independent losses like L_2 tend to produce blurry output, reflecting an averaged expectation instead of something realistic. Some have used generative adversarial networks (GANs) to mitigate this (Azadi et al., 2018), but these can suffer from missing modes and collapse issues. We instead introduce a novel adaptive loss to font reconstruction that operates on a wavelet image representation, while still permitting a well formed likelihood.

Specifically, in this paper we make the following contributions: (1) Propose the "Dual Manifold" model which treats both style and structure representations as latent variables (2) Propose a new adaptive loss function for synthesizing glyphs, and demonstrate its improvements over more common losses (3) Put forward two datasets that emphasize few-shot reconstruction, along with a preprocessing technique to remove near-duplicate fonts resulting in more challenging train/test splits.

We evaluate on the task of few-shot font reconstruction, reporting the structural similarity (SSIM) – a popular metric for image synthesis better correlated with human judgement than L_2 (Snell et al., 2017) – between reconstructions and a gold reference. These experiments are further split into *known* characters, which the model observed in at least one font at train time, and *unknown* characters, which can be thought of as a few-shot task. In addition we also perform human evaluation using Amazon Mechanical Turk. Our approach outperforms various baselines including nearest neighbor, the single manifold approach we build on (Srivatsan et al., 2019), and the previously mentioned discriminative model (Zhang et al., 2018).

2 Related Work

A variety of style transfer work has focused specifically on font style, and therefore, font reconstruction. Some approaches have sought to model the style aspect as a transformation on an underlying topological or stroke-based representation which must be learned for each character (Campbell and Kautz, 2014; Phan et al., 2015; Suveeranont and Igarashi, 2010). However this requires characters to have consistent topologies across fonts. Other work has learned a font skeleton manifold using Gaussian Process Latent Variable Models (Lian et al., 2018). One of the more philosophically similar approaches to ours is the bilinear factorization model of Freeman and Tenenbaum (Freeman and Tenenbaum, 1997; Tenenbaum and Freeman, 2000) which also learns vector representations for each font and character type, albeit in a non-probabilistic and linear manner. Some more recent research has treated font reconstruction as a discriminative task, using modern neural architectures and techniques from the style transfer literature (Zhang et al., 2018, 2020; Azadi et al.,

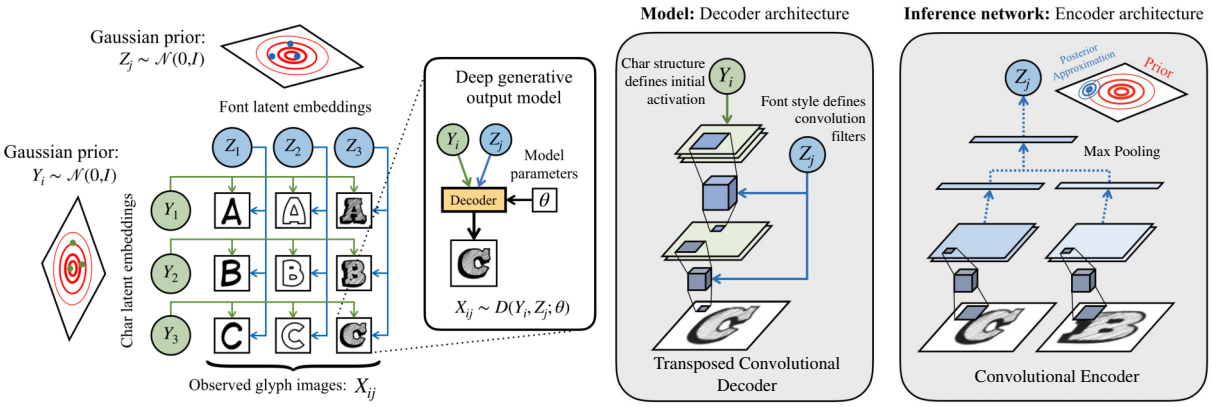


Figure 2: Overview of our generative factorization model, and important architecture details. Glyphs in the same row share a latent variable Y_i representing character shape, and those in the same column share Z_j representing font style. These variables are inferred by a network that takes in an entire row or column. Our decoder combines these representations to output a distribution on the glyph image.

2018; Gao et al., 2019). Furthermore the concept of learning manifolds for Chinese characters based on shared structure has also been studied (Cao et al., 2018), albeit with different downstream goals. Lopes et al. (2019) used VAEs which do not observe font alignment across glyphs, but condition on the character type (this work also primarily focuses on generating vector instead of pixel representations). Finally, more general-purpose style transfer methods for images are well explored (Gatys et al., 2015; Yang et al., 2019; Johnson et al., 2016; Wang and Gupta, 2016; Kazemi et al., 2019; Chen et al., 2017; Ulyanov et al., 2016a), although these largely lack inductive biases specially suited to typography.

3 Dual Manifold Model

Srivatsan et al. (2019) is the most similar prior work, as it also builds from a matrix factorization framework, and learns a latent manifold over font embeddings. Our model generalizes theirs by learning a second manifold over character shape, letting us massively scale up the number of characters that can be modeled. In Section 4 we also describe our novel loss.

Figure 2 depicts our model's generative process. For a corpus consisting of J fonts, each defined over up to I character types, we characterize each particular glyph image as a combination of properties relating to the style of that particular font and to the shape of that character. Our model effectively factorizes the data by assigning a vector representation to every row and column which correspond to character and font respectively. Therefore, our approach works by leveraging the fact that all glyphs of the same character type (*i.e.* an entire row in our data) share the same underlying structural shape, and all glyphs within the same font (*i.e.* an entire column) share the same stylistic properties. By forming separate representations over each of these two axes of variation, we can reconstruct missing glyphs in our data by separately inferring the relevant row and column variables, and then pushing new combinations of those inferred variables through our generative process. This can be thought of as a form of matrix completion, where unobserved entries correspond to characters not supported by particular fonts.

Given a corpus X consisting of I characters across J fonts, we assign to each observed glyph X_{ij} a pair of latent variables which model the properties of that glyph's character type and font style. Specifically we define these as $Y_i \in \mathbb{R}^k$ and $Z_j \in \mathbb{R}^k$, which we draw from a standard Gaussian prior $\mathcal{N}(0, I_k)$, with Y modeling the shape of the character (e.g. a `q` or `<`), and Z modeling the properties of the font (e.g. *Times New Roman* or *Roboto Light Italic*). Given a particular Y_i and Z_j, we combine them via a neural decoder to obtain a distribution $p(X_{ij}|Y_i, Z_j; \theta)$ which scores the corresponding glyph image X_{ij}. This yields the following likelihood function:

$$p(X, Y, Z; \theta) = \prod_I p(Y_i) \prod_J p(Z_j) \prod_{I,J} p(X_{ij}|Y_i, Z_j; \theta)$$

Both Y and Z are unobserved, and we must therefore infer both to train our model and pro-

duce reconstructions at test time. Note that by contrast, Srivatsan et al. (2019) represents characters as fixed parameters, and must only perform inference over font representations. We use a pair of encoder networks to perform amortized inference, as depicted in Figure 3.

3.1 Decoder Architecture

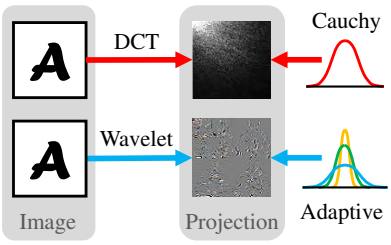
The basic structure of our decoder is largely identical to the popular U-Net architecture (Ronneberger et al., 2015) which has seen much success on image generation tasks with its coarse-to-fine layout of transposed convolutional layers. However, we make a few key modifications (depicted in Figure 2) in order to imbue our decoder with stronger inductive bias for this particular task. Following Srivatsan et al. (2019), instead of directly parameterizing the transposed convolutional layers that appear within each block of the network, we allow the weights of each layer to be the output of an MLP that takes as its input the font variable Z_j. This is effectively a form of HyperNetwork (Ha et al., 2016), a framework in which one network is used to produce the weights of another. In this way, the parameters of the transposed convolutional layers are dynamically chosen based on the font variable. By contrast, Y_i is the input fed in at the top of the decoder, to which these filters are applied. The purpose of this asymmetry is to encourage Z_j to learn properties relating to finer stylistic information, while Y_i learns more spatial information about the characters. In another manner of speaking, Y_i should learn "what" to write, and Z_j should learn "how" to write it.

4 Adaptive Wavelet Loss

Our decoder architecture outputs a grid of values, but an important decision is what distribution (and therefore loss) these should finally parameterize to score actual pixels. Traditional approaches using variational autoencoders have modeled each pixel as an independent Normal distribution, which results in the model minimizing the L_2 loss between its output and gold. This however leads to oversmoothed images, as it treats adjacent pixels as independent despite their strong correlations (Bell and Sejnowski, 1997), and fails to account for the heavy-tailed distribution of oriented edges in natural images (Field, 1987). As a result L_2 penalizes the model for generating images that are realistic but slightly transposed or otherwise not perfectly aligned with gold, which encourages models to produce fuzzy edges in order to be closer on average. GANs are often employed to force sharper output (Azadi et al., 2018; Gao et al., 2019), but following recent work we instead use a projected loss for a similar effect.

At a high level, our approach will first project images to a feature space, and let the model's output parameterize a distribution on this projection. If that projection is invertible and volume-preserving, this is equivalent to directly parameterizing a distribution on pixels, but allows for more expressivity (Rezende and Mohamed, 2015; Dinh et al., 2014, 2016). Ideally, such a loss requires a distribution expressive enough to capture the variable frequency characteristics of natural images, and a representation of the image that explicitly reasons about spatially-localized edges.

A good example of this technique is that of Srivatsan et al. (2019), which modeled images by placing a Cauchy distribution on a 2D Discrete Cosine Transform (DCT) representation of glyphs. Though this is an improvement over the default choice of placing a Normal distribution on individual pixels as it both decorrelates pixels and is tolerant of outliers, this approach is limited in its expressiveness and its ability to model spatially localized edges: Cauchy distributions are excessively heavy-tailed and so have difficulty modeling inliers, and since DCT is a global representation it does not allow the model to reason about *where* image gradient content is located.



We extend this approach in two ways (as depicted above): (1) by using a wavelet image representation instead of DCT, and (2) by using a distribution with an adaptive shape instead of a Cauchy.

Representation We opt for a wavelet representation, as unlike DCT it jointly encodes the frequency *and spatial location* of an image feature. As might be expected, an image representation in which location is directly encoded is helpful in our task; a stroke has a fundamentally different meaning at the top of a character than at the bottom. Barron (2019) quantitatively demonstrated the advantages of specifically the Cohen-Daubechies-Feauveau (CDF) 9/7 wavelet decomposition (Cohen et al., 1992) for training likelihood-based models of natural images. Based on their findings that CDF 9/7 in front of an adaptive loss achieves better performance than DCT in front of a Cauchy (the setup of Srivatsan et al. (2019)), we expect similar performance benefits in the context of our own model, and our ablations in Table 1 (Right) support this belief empirically.

Distribution An adaptive distribution lets the model select between using leptokurtotic (Cauchy-like) distributions that are well suited to the high-frequency image edges found at the finer levels of the wavelet decomposition, or more platykurtic (Normal-like) distributions that are better suited to low-frequency DC-like average image intensities found at the coarsest levels of the wavelet decomposition. Specifically, we use the probability distribution of Barron (2019):

$$f(x|\mu,\sigma,\alpha) = \frac{\exp\left(-\frac{|\alpha-2|}{\alpha}\left(\left(\frac{(x-\mu)^2}{\sigma^2|\alpha-2|}+1\right)^{\alpha/2}-1\right)\right)}{\sigma Z(\alpha)}$$

where $Z(\alpha)$ is the distribution's partition function, and α determines the distribution's shape. As $\alpha \to 0$ the distribution approaches a Cauchy distribution, as $\alpha \to 2$ the distribution approaches a Normal distribution.

Taken together, these yield a conditional likelihood function parameterized by the decoder of our variational model, which we now describe. Given an image $X_{i,j}$, we first project it using the CDF 9/7 wavelet decomposition – which we denote as $\psi(X_{i,j})$. Because this decomposition is a biorthogonal volume-preserving transformation, it can be applied before the likelihood computation. It further serves as a whitening transformation, avoiding the need to learn a covariance matrix for $X_{i,j}$.

Our decoder outputs a grid of parameters $\hat{X}_{i,j}$, the projection of which serves as the mean μ of our adaptive distribution for scoring $\psi(X_{i,j})$. For the other distribution parameters σ and α, rather than using fixed settings we construct a set of latent variables for both: we allow each wavelet coefficient to have its own vector of latent shape parameter $\boldsymbol{\ell}^{\alpha}$ and scale parameter $\boldsymbol{\ell}^{\sigma}$, where the non-latent shape and scale are parameterized as scaled and shifted sigmoids and softplus of those latent values:

$$\alpha_k = \frac{2}{1+\exp\left(\ell_k^{\alpha}\right)}, \sigma_k = \frac{\log\left(1+\exp\left(\ell_k^{\alpha}\right)\right)}{\log(2)} + \epsilon$$

We initialize $\boldsymbol{\ell}^{\alpha} = \boldsymbol{\ell}^{\sigma} = \vec{0}$, thereby initializing $\boldsymbol{\alpha} = \boldsymbol{\sigma} = \vec{1}$. These latent variables $(\boldsymbol{\ell}^{\alpha}, \boldsymbol{\ell}^{\sigma})$ are optimized during training using gradient descent along with all other model parameters θ, which allows the model to adapt the shape and scale of each wavelet coefficient's distribution during training. Overall, this yields the following likelihood function:

$$p(X_{i,j}|Y_i, Z_j; \theta) = \prod_k f(\psi(X_{i,j})_k | \psi(\hat{X}_{i,j})_k, \sigma_k, \alpha_k)$$

5 Learning and Inference

We now describe our approach to training this model. This process mirrors that of previous variational work, although since we are learning a dual manifold, our model will require two separate inference networks. The projected loss we add (Section 4) will not fundamentally affect the learning process, but does change how the reconstruction term is computed.

As our model is generative, we wish to maximize the log likelihood of the training data with respect to the model parameters, which requires summing out the unobserved variables Y and Z. However, this integral is intractable and does not permit a closed form solution. We therefore resort to optimizing a variational approximation, a strategy which has seen success in similar settings (Kingma and Welling, 2014; Srivatsan et al., 2019). Rather than directly optimize the likelihood (which we cannot compute the gradient of), we maximize a lower bound on it known as the Evidence Lower Bound (ELBO). We compute the ELBO via a function $q(Y, Z|X) = q(Y|X) * q(Z|Y, X)$ which approximates the posterior $p(Z, Y|X)$ of the distribution defined by our decoder network.

$$\text{ELBO} = \mathbb{E}_q[\log p(X|Z,Y)] - \mathbb{KL}(q(Z,Y|X)||p(Z)p(Y))$$

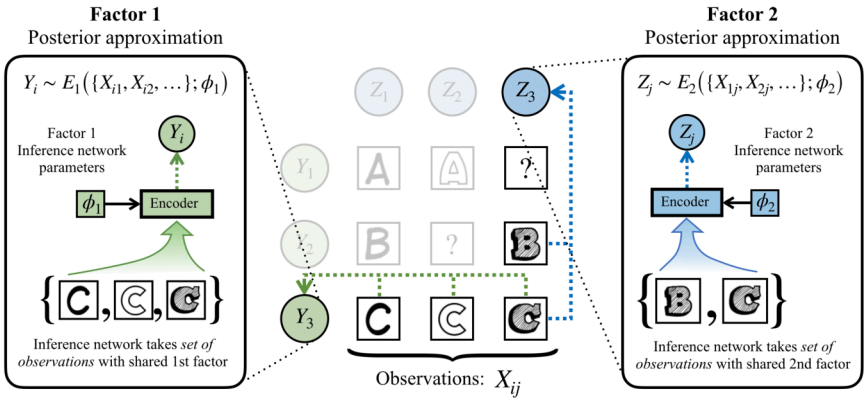


Figure 3: Overview of the inference procedure. First the character encoder infers a representation of structure over each row, and then the font encoder infers a representation of style conditioned on a (perhaps partially observed) column and the character embeddings.

We define $q(Y|X)$ and $q(Z|Y, X)$ via a pair of encoder networks which operate over one row or column of the matrix respectively. An encoder passes each glyph in that row or column through a series of convolutional layers, and then max pools the output features across all glyphs, ensuring it can handle a variable number of observations (See Figure 3). Note that the method of pooling (*e.g.* min, max, avg), as well as the order in which to infer Y and Z are important choice points that allow for different inductive biases. The pooled feature representation is then passed through an MLP which outputs parameters μ and Σ to define a Gaussian posterior over Y_i or Z_j. Given these, we compute approximate gradients on the ELBO via the reparameterization trick described by Kingma and Welling (2014).

6 Experiments

We evaluate on the task of font reconstruction, in which given a small random subset of glyphs from a held out font, models must reconstruct the remaining ones. We separately report performance on known (*i.e.* observed at least once during training) and unknown character types. During training, we mask out a randomly chosen 20% of character *types* to serve as unknowns. At test time, models observe examples of previously masked characters to infer their representations for reconstruction. This can be thought of as a few-shot task, where models must generate glyphs for character types they did not observe at train time based on limited test-time examples.

6.1 Datasets

Capitals64, the dataset used by Azadi et al. (2018) and Srivatsan et al. (2019), only contains the 26 English capital letters, with no missing characters, meaning it does not require learning a manifold over character shape. We instead evaluate on the following datasets to best demonstrate our method's ability to scale to settings with a large number of character types and a high degree of sparsity.

Google Fonts Google Fonts is a dataset of 991 font families, which is publicly available[1]. Most fonts in the dataset support standard Latin characters, but many also support special symbols, and characters found in Greek, Cyrillic, Tamil, and several other orthographies. A visualization of this is shown in Figure 1. We restrict our work to the 2000 most frequently supported character types for simplicity. After removing near duplicates (described below) we are left with 2017 fonts in total, split into train, dev, and test in a $3:1:1$ ratio. The data was split by font family rather than individual fonts, to ensure that there are no fonts in train with a "sibling" in test.

Chinese Simplified We scraped a list of the most common 2000 Chinese simplified characters from the internet as well as a dataset that labels each character's radical. Together, we compile a new dataset that consists of the most common 2000 Chinese characters along with their radicals for further analysis on the character embeddings. For each Chinese character, we scraped over 1524 fonts, split similarly to Google Fonts. The total font number shrinks down to 623 after removing near duplicates, which we now discuss.

Removing near duplicates One major issue with font corpora is that most fonts belong to a small handful of modes, within which there is little stylistic diversity. To ensure

[1] https://github.com/google/fonts

that our metrics best measure generalization to novel fonts unlike those seen in train, we preprocess out fonts that are extremely similar to others in the data. We first perform agglomerative clustering, and then retain only the centroid of each cluster. The number of clusters is determined by cutting the dendrogram at a height which eliminates most fonts that are to a human largely indistinguishable from their nearest neighbor.

6.2 Baselines

We compare our model – which we refer to as DUAL MANIFOLD – to two baselines and various ablations. Our primary baseline is EMD (Zhang et al., 2018), a discriminative encoder-decoder model that does not share embeddings across "rows" and "columns", but rather computes style and content representations for each glyph given a set of provided examples, and then passes them to a generator which constructs the final image. This model is useful for comparison as it has a similar computation graph and also learns separate embeddings for font and character shape, but computes its loss directly in pixel space, and lacks a probabilistic prior.

We also compare to a naive nearest neighbor (NN) model, which reconstructs fonts at test time by finding the font in train with the closest L_2 distance over the observed characters, and outputs that neighbor's corresponding glyphs for the missing characters. If the neighbor does not support all missing characters, we pull the remaining from the 2nd nearest neighbor, and so on. It should be noted that NN cannot reconstruct any character that is not present in train.

Similarly to EMD, the first of our ablations, denoted -KL, does not explicitly model the character and font embeddings as random variables. This effectively removes the KL divergence from the loss function, resulting in a non-probabilistic autoencoder. The next, denoted -DUAL, is an ablation which treats the character representations as parameters of the model, rather than latent variables which must be inferred. This is essentially the model of Srivatsan et al. (2019) with our architecture*. We also ablate our adaptive wavelet loss against the DCT + Cauchy loss used by Srivatsan et al. (2019), denoted with -ADAPT. Finally, we compare performing MAX or MIN pooling over elements of a row/column within the encoder network.

6.3 Training Details

We perform stochastic gradient descent using Adam (Kingma and Ba, 2015), with a step size of 10^{-4}. Batches contain 10 fonts, each with only 20 random characters observable to encourage robustness to the number of inputs. However at test time, the model can infer the character representations Y based on the the entire training set. We find best results when both character and style representations are $k = 256$ dimensional. See Appendix A for a full description of architecture. Our model trains on one NVIDIA 2080ti GPU in roughly a week, and is implemented in PyTorch (Paszke et al., 2017) version `1.3.1`

6.4 Metrics

We measure average SSIM per glyph (Azadi et al., 2018; Gao et al., 2019), having scaled pixel intensities to $[0, 1]$. While the details of SSIM are beyond the scope of this paper, it can be thought of as a feature-based metric that does not factor over individual pixels, but rather looks at the matches between higher level features regarding the structure of the image. SSIM is widely used in image processing tasks since it measures structural similarity instead of raw pixel distance, and has been shown to better correlate with human judgement than L_2 (Snell et al., 2017). Evaluating models using L_2 can reward unrealistic reconstructions that split the difference between many hypotheses as opposed to picking just one (part of the reason we avoid training our model on such a loss). Over the course of individual training runs, we found it was almost counter correlated with human judgement, with the lowest distance early in training while output was blurry, becoming larger as the model converged. We do however include these numbers in Appendix B, as they nonetheless support our findings.

We also perform human evaluation using Amazon Mechanical Turk. For each font in our test set, 5 turkers were shown 8 example glyphs, and a sample of reconstructions for the remaining characters by DUAL MANIFOLD and EMD. Turkers were also shown examples of

Observations	1	8	16	32	1	8	16	32
	Google Fonts: Known Char				Google Fonts: Unknown Char			
NN	0.755	0.816	0.830	**0.839**	-	-	-	-
EMD	0.706	0.702	0.539	0.597	0.698	0.695	0.534	0.595
Dual Manifold	**0.799**	**0.828**	**0.833**	0.834	**0.801**	**0.826**	**0.829**	**0.830**
	Chinese Simplified: Known Char				Chinese Simplified: Unknown Char			
NN	**0.428**	**0.488**	**0.495**	**0.499**	-	-	-	-
EMD	0.278	0.271	0.291	0.288	0.270	0.266	0.283	0.280
Dual Manifold	0.392	0.405	0.407	0.407	**0.375**	**0.387**	**0.390**	**0.390**

Observations	1	8	16	32
+Dual, +KL, +Adapt, Min	0.7728	0.8041	0.8083	0.8088
Srivatsan et al. (2019)*	0.713	0.702	0.701	0.698
-Dual, +KL, +Adapt, Max	0.704	0.703	0.701	0.703
+Dual, -KL, +Adapt, Max	0.785	0.817	0.821	0.823
+Dual, +KL, -Adapt, Max	0.795	0.823	0.828	0.829
+Dual, +KL, +Adapt, Max	**0.799**	**0.828**	**0.833**	**0.834**

Table 1: (Left) SSIM per glyph by number of observed characters for Google Fonts and Chinese Simplified. (Right) Ablations of our model, showing SSIM results on known characters in Google Fonts.

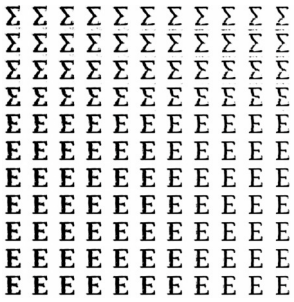


Figure 4: Generated glyphs for interpolating between both character type (horizontal axis) and font style (vertical axis) simultaneously.

each character in a neutral style. They were asked to select which if either reconstruction was better, and briefly justify their reasoning.

7 Results

7.1 Quantitative Evaluation

We list SSIM results in Table 1 for various numbers of observed characters. Note that NN is not capable of reconstructing character types not observed at training time. On Google Fonts, our model performs best overall; however on Chinese Simplified, we see NN winning on known characters, as well as a marked drop in SSIM overall. This could be due to the increased challenge in generating Chinese characters given the relatively higher number of strokes, leading SSIM to prefer the realism of NN, or because fonts in this dataset generally contain most characters, unlike Google Fonts which is much sparser. Observing more characters taperingly increases similarity, which matches our intuition that this allows for a better understanding of stylistic properties. Performance drops when evaluating on characters not observed in training. This makes sense as models may have less support in their manifolds for structural forms they were not trained on, but the drop is small enough to suggest our model is able to infer meaningful representations for novel character types at test time.

We see also that EMD has significant issues

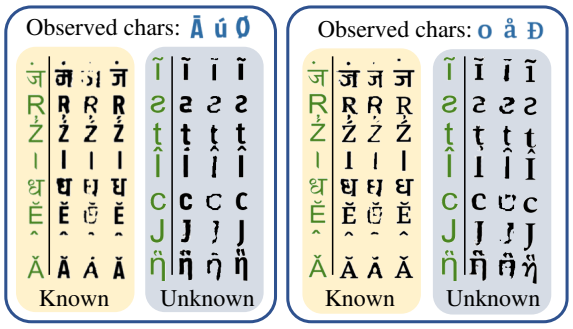


Figure 5: Reconstructions of two fonts from our model, EMD, and NN — shown in black in that order — for both known and unknown character types. Green characters show the expected shape in a neutral font, and blue characters are a sample of those observed by the models for either font.

at 16 and 32 observed characters (it's worth noting that EMD must be separately trained for each number of observations). Qualitatively, we find certain fonts for which EMD emits the same output for every character in that font. We suspect this indicates overfitting leading to broken style representations for some novel fonts when given more observations than its default of only 10.

Within our ablations, we find that using a dual latent manifold, as opposed to treating character embeddings as model parameters, is responsible for the majority of our gain in SSIM over prior work. The next largest difference comes from using either Min pooling within the autoencoder or Max pooling. We also see more of a drop in performance from removing the KL divergence, than we do from replacing our adaptive wavelet loss with the DCT + Cauchy loss.

7.2 Human Evaluation

In our AMT experiments, we found that for **48.2%** of known character reconstructions, turkers preferred our model's output, with

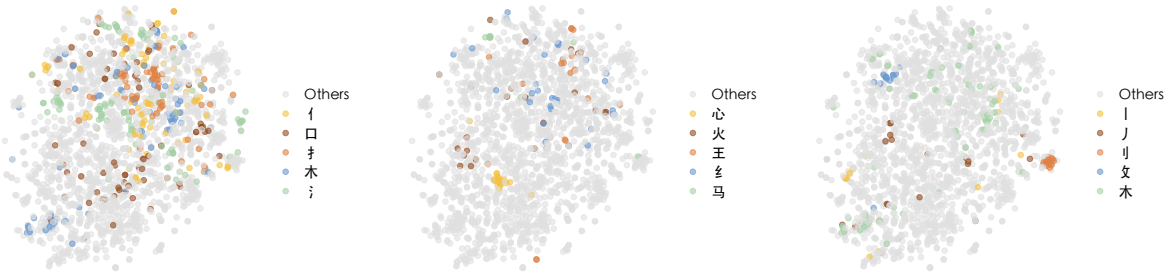


Figure 6: t-SNE plot of Chinese character embeddings from our model for the top 5 radicals (left), and randomly chosen groups of 5 (middle, right).

42.0% preferring EMD, and 9.8% finding both equal. For unknown character reconstructions, **50.5%** preferred ours, vs 38.7% for EMD, and 10.9% finding no difference. A majority of turkers agreed 86.3% of the time in the case of known characters, and 83.2% for unknown.

8 Analysis

8.1 Qualitative Inspection

In Figure 4 we show output from our model interpolating between a bold font and a light one, as well as a capital E and a Σ simultaneously. This demonstrates the smoothness of our manifolds and also suggests how they might offer support for font and character types not seen during training. Figure 5 shows examples of reconstructions by models on two fonts for a variety of both known and unknown characters. Our approach is more coherent and faithful than EMD, and NN is realistic but often stylistically incorrect.

8.2 Chinese Radicals

Figure 6 shows t-SNE projections (Maaten and Hinton, 2008; Van Der Maaten, 2014) of learned character embeddings colored by their radical, a sub-component of Chinese characters. Radicals like 丨, 丿, and 火 — which either share forms with others or can occur in different structures — don't cluster together, while unique radicals like 心, 刂, and 攵 do.

9 Conclusion

In this work introduced a generative model for typography capable of reconstructing characters in a novel font, of a novel shape, or both, and demonstrated its improvements over previous approaches on two datasets containing large numbers of characters. We analyzed the results qualitatively, and inspected learned manifolds for smoothness.

In future work, this methodology has potential value not just to fonts, but to any domain which can also be factored over independent axes of variation, such as handwriting by different authors. One could also incorporate this model into more complex downstream tasks such as OCR. That being said, these domains also feature complex interactions between physically adjacent glyphs (our model treats different characters within a font as conditionally independent), so some further innovation would likely still be required.

There are also extensions to the model itself that might be worth exploring in future work, for instance operating on a stroke-based representation in order to perform reconstruction in the original TTF space instead of raw pixel space as we do here. This would also likely assist with smoothness of edges and reduce the incidence of "corroded" output glyphs.

Broader Impact

As our work can be used to augment or even replace the labor of human artists, it is worth discussing its potential broader impacts. The most obvious positive is that this technique can add value to font designers, by minimizing the overhead required to design a font that supports widespread internationalization. Our model's ability to interpolate stylistic properties can also make it easy to automatically generate completely novel fonts that are roughly similar to existing ones.

This also benefits speakers of languages that rely on less common glyphs, as it broadens their font selection. It can make it easier for them to both produce and consume digital content, allowing for better accessibility for demo-

graphics that currently have fewer options for orthographies they are most familiar with.

One potential negative impact is on the business of some font artists who cater to niche audiences that have less common glyph needs. Our model could potentially be used to replace such workers, and if so could also lead to less coherent renderings for uncommon orthographies if those who are not fluent in such scripts simply employ our system without a thorough understanding of the types of errors it may make.

Acknowledgements

This project is funded in part by the NSF under grants 1618044 and 1936155, and by the NEH under grant HAA256044-17.

References

Samaneh Azadi, Matthew Fisher, Vladimir G Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell. 2018. Multi-content GAN for few-shot font style transfer. *CVPR*.

Jonathan T. Barron. 2019. A general and adaptive robust loss function. *CVPR*.

Anthony J Bell and Terrence J Sejnowski. 1997. Edges are the'independent components' of natural scenes. *NeurIPS*.

Neill DF Campbell and Jan Kautz. 2014. Learning a manifold of fonts. *ACM TOG*.

Shaosheng Cao, Wei Lu, Jun Zhou, and Xiaolong Li. 2018. cw2vec: Learning chinese word embeddings with stroke n-gram information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Dongdong Chen, Lu Yuan, Jing Liao, Nenghai Yu, and Gang Hua. 2017. Stylebank: An explicit representation for neural image style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1897–1906.

Albert Cohen, Ingrid Daubechies, and J-C Feauveau. 1992. Biorthogonal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, 45.

Laurent Dinh, David Krueger, and Yoshua Bengio. 2014. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2016. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

David J. Field. 1987. Relations between the statistics of natural images and the response properties of cortical cells. *JOSA-A*.

William T Freeman and Joshua B Tenenbaum. 1997. Learning bilinear models for two-factor problems in vision. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 554–560. IEEE.

Yue Gao, Yuan Guo, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. 2019. Artistic glyph image synthesis via one-stage few-shot learning. *ACM Transactions on Graphics (TOG)*, 38(6):1–12.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. 2015. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*.

Google. Google fonts. http://fonts.google.com. Accessed: 2020-6-4.

David Ha, Andrew Dai, and Quoc V Le. 2016. Hypernetworks. *arXiv preprint arXiv:1609.09106*.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer.

Hadi Kazemi, Seyed Mehdi Iranmanesh, and Nasser Nasrabadi. 2019. Style and content disentanglement in generative adversarial networks. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 848–856. IEEE.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *ICLR*.

Diederik P Kingma and Max Welling. 2014. Auto-encoding variational bayes. *ICLR*.

Zhouhui Lian, Bo Zhao, Xudong Chen, and Jianguo Xiao. 2018. Easyfont: a style learning-based system to easily build your large-scale handwriting fonts. *ACM Transactions on Graphics (TOG)*, 38(1):1–18.

Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. 2019. A learned representation for scalable vector graphics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7930–7939.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Huy Quoc Phan, Hongbo Fu, and Antoni B Chan. 2015. Flexyfont: Learning transferring rules for flexible typeface synthesis. In *Computer Graphics Forum*, volume 34, pages 245–256. Wiley Online Library.

Danilo Rezende and Shakir Mohamed. 2015. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. *MICCAI*.

Jake Snell, Karl Ridgeway, Renjie Liao, Brett D Roads, Michael C Mozer, and Richard S Zemel. 2017. Learning to generate images with perceptual similarity metrics. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 4277–4281. IEEE.

Nikita Srivatsan, Jonathan T. Barron, Dan Klein, and Taylor Berg-Kirkpatrick. 2019. A deep factorization of style and structure in fonts. *EMNLP*.

Rapee Suveeranont and Takeo Igarashi. 2010. Example-based automatic font generation. In *International Symposium on Smart Graphics*, pages 127–138. Springer.

Joshua B Tenenbaum and William T Freeman. 2000. Separating style and content with bilinear models. *Neural computation*, 12(6):1247–1283.

Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. 2016a. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, volume 1, page 4.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. 2016b. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*.

Unicode. Unicode® 13.0.0. http://unicode.org/versions/Unicode13.0.0/. Accessed: 2020-5-27.

Laurens Van Der Maaten. 2014. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245.

Xiaolong Wang and Abhinav Gupta. 2016. Generative image modeling using style and structure adversarial networks. In *European conference on computer vision*, pages 318–335. Springer.

Shuai Yang, Jiaying Liu, Wenjing Wang, and Zongming Guo. 2019. Tet-gan: Text effects transfer via stylization and destylization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1238–1245.

Richard Zhang. 2019. Making convolutional networks shift-invariant again. In *International Conference on Machine Learning*, pages 7324–7334. PMLR.

Yexun Zhang, Ya Zhang, and Wenbin Cai. 2018. Separating style and content for generalized style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8447–8455.

Yexun Zhang, Ya Zhang, and Wenbin Cai. 2020. A unified framework for generalizable style transfer: Style and content separation. *IEEE Transactions on Image Processing*, 29:4085–4098.

A Architecture Details

The architectures of our encoder and decoder are largely identical to that of U-Net (Ronneberger et al., 2015) with key differences described here. We find significantly improved results by inserting Instance Normalization layers (Ulyanov et al., 2016b) after convolution layers in our decoder. We also replace the max pool layers within the encoder with blur pool layers (Zhang, 2019). As stated previously, we max pool the output of the encoders across character types or fonts, and then pass the flattened pooled representation through a fully connected layer to obtain the approximate posterior parameters μ and Σ. A similar fully connected layer projects the character representation Y_i to the appropriate size before being passed to the decoder. As noted earlier, the parameters of the last two transposed convolutional layers in the decoder are dynamically output by MLPs which take as input the font representation Z_j. These consist of a 256 dimensional fully connected layer, a ReLU, and then a second fully connected layer to produce the relevant parameter.

We now provide further details on the specific layer sizes used in our model and inference network. The following abbreviations are used to represent various components:

- F_i : fully connected layer with i hidden units
- R : ReLU activation
- S : sigmoid activation
- M : batch max pool
- B : 2×2 spatial blur pool (Zhang, 2019)
- C_i : convolutional layer with i filters of 3×3, 1 pixel zero-padding, stride of 1
- I : instance normalization
- T_i : transpose convolution with i filters of 2×2, stride of 2
- D_i : transpose convolution with i filters of 2×2, stride of 2, where kernel and bias are the output of an MLP (described below)
- H : reshape to $-1 \times 256 \times 8 \times 8$

Our encoder is:

$C_{64} - R - C_{64} - R - C_{64} - R - B - C_{128} - R - C_{128} - R - B - C_{256} - R - C_{256} - R - B - C_{512} - R - C_{512} - R - B - M - F_{512}$

Our decoder is:

$F_{1024 \times 8 \times 8} - T_{1024} - C_{512} - I - R - C_{512} - I - R - T_{512} - C_{256} - I - R - C_{256} - I - R - D_{256} - C_{128} - I - R - C_{128} - I - R - D_{128} - C_{64} - I - R - C_{64} - I - R - C_1 - S$

MLP to compute transpose convolutional parameter of size j is:

$F_{256} - R - F_j$

B L_2 Results

In Table 2 we show results on Google Fonts and Chinese simplified for our model and baselines in terms of L_2. Rankings are generally the same, and see that our approach performs best by this metric as well as SSIM. We do however note that in places the L_2 numbers and SSIM numbers are not well correlated, and attribute this to L_2's propensity for rewarding blurry output that minimizes expected distance over sharp output that may have slightly misaligned edges.

	Google Fonts: Known Char				Google Fonts: Unknown Char			
Observations	1	8	16	32	1	8	16	32
NN	405.15	258.11	227.91	207.04	-	-	-	-
EMD	371.06	367.18	658.85	512.26	378.08	375.19	667.66	511.69
Dual Manifold	**275.56**	**202.58**	**193.34**	**189.94**	**276.47**	**212.76**	**205.34**	**202.91**
	Chinese Simplified: Known Char				Chinese Simplified: Unknown Char			
NN	1086.58	908.58	883.18	872.52	-	-	-	-
EMD	1013.80	1019.97	1288.32	1287.85	1303.96	1020.89	1303.92	1303.48
Dual Manifold	**916.41**	**879.03**	**873.48**	**868.41**	**917.91**	**883.60**	**878.77**	**875.03**

Table 2: L_2 per glyph by number of observed characters for Google Fonts and Chinese Simplified.