A Learned Representation for Scalable Vector Graphics

Raphael Gontijo Lopes,[*] David Ha, Douglas Eck, Jonathon Shlens
Google Brain
{iraphael, hadavid, deck, shlens}@google.com

Abstract

Dramatic advances in generative models have resulted in near photographic quality for artificially rendered faces, animals and other objects in the natural world. In spite of such advances, a higher level understanding of vision and imagery does not arise from exhaustively modeling an object, but instead identifying higher-level attributes that best summarize the aspects of an object. In this work we attempt to model the drawing process of fonts by building sequential generative models of vector graphics. This model has the benefit of providing a scale-invariant representation for imagery whose latent representation may be systematically manipulated and exploited to perform style propagation. We demonstrate these results on a large dataset of fonts and highlight how such a model captures the statistical dependencies and richness of this dataset. We envision that our model can find use as a tool for graphic designers to facilitate font design.

1. Introduction

The last few years have witnessed dramatic advances in generative models of images that produce near photographic quality imagery of human faces, animals, and natural objects [4, 25, 27]. These models provide an exhaustive characterization of natural image statistics [52] and represent a significant advance in this domain. However, these advances in image synthesis ignore an important facet of how humans interpret raw visual information [48], namely that humans seem to exploit *structured* representations of visual concepts [33, 21]. Structured representations may be readily employed to aid generalization and efficient learning by identifying higher level primitives for conveying visual information [32] or provide building blocks for creative exploration [21, 20]. This may be best seen in human drawing, where techniques such as *gesture drawing* [44] emphasize parsimony for capturing higher level semantics and actions with minimal graphical content [54].

[*]Work done as a member of the Google AI Residency Program (`g.co/airesidency`)

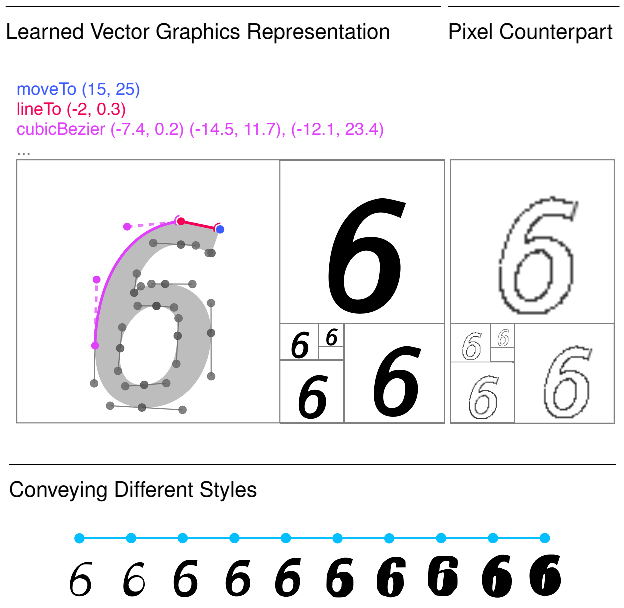

Learned Vector Graphics Representation
Pixel Counterpart
moveTo (15, 25)
lineTo (-2, 0.3)
cubicBezier (-7.4, 0.2) (-14.5, 11.7), (-12.1, 23.4)
...
Conveying Different Styles

Figure 1: **Learning fonts in a native command space**. Unlike pixels, scalable vector graphics (SVG) [11] are scale-invariant representations whose parameterizations may be systematically adjusted to convey different styles. All vector images are samples from a generative model of the SVG specification.

Our goal is to train a drawing model by presenting it with a large set of example images [16, 13]. To succeed, the model needs to learn both the underlying structure in those images and to generate drawings based on the learned representation [2]. In computer vision this is referred to as an “inverse graphics” problem [38, 31, 22, 41]. In our case the output representation is not pixels but rather a sequence of discrete instructions for rendering a drawing on a graphics engine. This poses dual challenges in terms of learning discrete representations for latent variables [57, 23, 37] and performing optimization through a non-differential graphics engine (but see [36, 31]). Previous approaches focused on program synthesis approaches [32, 10] or employing reinforcement and adversarial learning [13]. We instead focus on a subset of this domain where we think we can make progress and improves the generality of the approach.

Font generation represents a 30 year old problem posited as a constrained but diverse domain for understanding higher level perception and creativity [21]. Early research attempted to heuristically systematize the creation of fonts for expressing the identity of characters (e.g. a, 2) as well as stylistic elements constituting the "spirit" of a font [20]. Although such work provides great inspiration, the results were limited by a reliance on heuristics and a lack of a learned, structured representation [47]. Subsequent work for learning representations for fonts focused on models with simple parameterizations [34], template matching [55], example-based hints [63], or more recently, learning manifolds for detailed geometric annotations [5].

We instead focus the problem on generating fonts specified with Scalable Vector Graphics (SVG) – a common file format for fonts, human drawings, designs and illustrations [11]. SVG's are a compact, scale-invariant representation that may be rendered on most web browsers. SVG's specify an illustration as a sequence of a higher-level commands paired with numerical arguments (Figure 1, top). We take inspiration from the literature on generative models of images in rasterized pixel space [15, 56]. Such models provide powerful auto-regressive formulations for discrete, sequential data [15, 56] and may be applied to rasterized renderings of drawings [16]. We extend these approaches to the generation of sequences of SVG commands for the inference of individual font characters.

The goal of this work is to build a tool to learn a representation for font characters and style that may be extended to other artistic domains [7, 50, 16], or exploited as an intelligent assistant for font creation [6]. We aspire that our methods could be applied generically, but we focus on font generation as our main inspiration in hopes that it opens opportunities to work on more complex illustrations [7]. To this end, our main contributions are as follows:

- Build a generative model for scalable vector graphics (SVG) images and apply this to a large-scale dataset of 14 M font characters.
- Demonstrate that the generative model provides a perceptually smooth latent representation for font styles that captures a large amount of diversity and is consistent across individual characters.
- Exploit the latent representation from the model to infer complete SVG fontsets from a single (or multiple) characters of a font.
- Identify semantically meaningful directions in the latent representation to globally manipulate font style.

2. Related Work

2.1. Generative models of images

Generative models of images have generally followed two distinct directions. Generative adversarial networks [14] have demonstrated impressive advances [46, 14] over the last few years resulting in models that generate high resolution imagery nearly indistinguishable from real photographs [25, 4].

A second direction has pursued building probabilistic models largely focused on invertible representations [8, 27]. Such models are highly tractable and do not suffer from training instabilities largely attributable to saddle-point optimization [14]. Additionally, such models afford a true probabilistic model in which the quality of the model may be measured with well-characterized objectives such as log-likelihood.

2.2. Autoregressive generative models

One method for vastly improving the quality of generative models with unsupervised objectives is to break the problem of joint prediction into a conditional, sequential prediction task. Each step of the conditional prediction task may be expressed with a sequential model (e.g. [19]) trained in an autoregressive fashion. Such models are often trained with a teacher-forcing training strategy, but more sophisticated methods may be employed [1].

Autoregressive models have demonstrated great success in speech synthesis [42] and unsupervised learning tasks [57] across multiple domains. Variants of autoregressive models paired with more sophisticated density modeling [3] have been employed for sequentially generating handwriting [15].

2.3. Modeling higher level languages

The task of learning an algorithm from examples has been widely studied. Lines of work vary from directly modeling computation [24] to learning a hierarchical composition of given computational primitives [12]. Of particular relevance are efforts that learn to infer graphics programs from the visual features they render, often using constructs like variable bindings, loops, or simple conditionals [10].

The most comparable methods to this work yield impressive results on unsupervised induction of programs usable by a given graphics engine [13]. As their setup is non differentiable, they use the REINFORCE [60] algorithm to perform adversarial training [14]. This method achieves impressive results despite not relying on labelled paired data. However, it tends to draw over previously-generated drawings, especially later in the generation process. While this could be suitable for modelling the generation of a 32x32 rastered image, SVGs require a certain level of precision in order to scale with few perceptible issues.

Modelling languages for image generation [13, 16], which our work tackles, can be construed as a probabilistic programming problem. What makes our problem unique, however, is (a) models must be perceptually judged [13] and (b) by working in the SVG format, we open the opportunity to exploit a de-facto standard format for icons and graphics.

2.4. Learning representations for fonts

Previous work has focused on enabling propagation of style between classes by identifying the class and style from high level features of a single character [47, 21, 20] or by finding correspondences between these features of different characters [55]. These features are often simplifications, such as character skeletons, which diminish the flexibility of the methods. Other work directly tackles style manipulation of generated full characters [34], but uses a simple parametric model that allows the user to only adjust parameters like its weight or width. Finally, parallel work use a generative model of rasterized fonts for augmenting handwriting classification in low-resource settings [45].

The most relevant works are those that attempt to learn a manifold of font style. Some unpublished work has explored how probabilistic methods may model pixel-based representations of font style [35]. The model learned semantically-meaningful latent spaces which can manipulate rasterized font images. More directly comparable, recent work learned energy models to capture the relationship between discretized points along the outlines of each character in order to address font generation and extrapolation [5]. This method yields impressive results for extrapolating between very few examples, but is limited by the need of having all characters of a certain class be composed of equivalent shapes Additionally, the model discretely approximates points on a character's outline which may lead to visual artifacts at larger spatial scales.

3. Methods

3.1. Data

We compiled a font dataset composed of 14 M examples across 62 characters (i.e. `0-9`, `a-z`, `A-Z`), which we term `SVG-Fonts`. The dataset consists of fonts in a common font format (SFD) [1], excluding examples where the unicode ID does not match the targeted 62 character set specified above. In spite of the filtering, label noise exists across the roughly 220 K fonts examined.

We employed a one-to-one mapping from SFD to SVG file format using a subset of 4 SVG commands: `moveTo`, `lineTo`, `cubicBezier` and `EOS`. SVG commands were normalized by starting from the top-most command and ordering in a clockwise direction. In preliminary experiments, we found it advantageous to specify command arguments

[1]`https://fontforge.github.io`

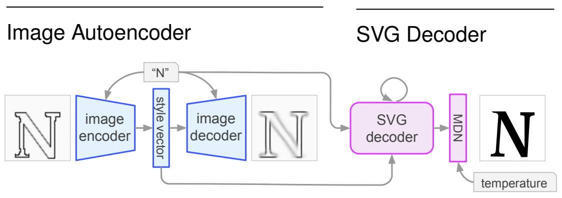


Figure 2: **Model architecture**. Visual similarity between SVGs is learned by a class-conditioned, convolutional variational autoencoder (VAE) [30, 16] on a rendered representation (blue). The class label and learned representation z are provided as input to a model that decodes SVG commands (purple). The SVG decoder consists of stacked LSTM's [19] followed by a Mixture Density Network (MDN) [15, 3]. See text for details.

using relative positioning information. See Appendix for more details of the dataset collection and normalization.

The final dataset consists of a sequence of commands specified in tuples. Each item in the sequence consists of a discrete selection of an SVG command paired with a set of normalized, floating-point numbers specifying command arguments. We restricted the dataset to only 4 SVG command types and examples with fewer then 50 commands to aid learning but these restrictions may be relaxed to represent the complete SVG language. In comparison, note that [13] restricted inference to 20 actions for generating images. Finally, we partitioned the dataset into 12.6M and 1.4M examples for training and testing [2]

3.2. Network Architecture

The model consists of a variational autoencoder (VAE) [30, 16] and an autoregressive SVG decoder implemented in Tensor2Tensor [58]. Figure 2 provides a diagram of the architecture but please see the Appendix for details. Briefly, the VAE consists of a convolutional encoder and decoder paired with instance normalization conditioned on the label (e.g. `a`, `2`, etc.) [9, 43]. The VAE is trained as a class-conditioned autoencoder resulting in a latent code z that is largely class-independent [28]. In preliminary experiments, we found that 32 dimensions provides a reasonable balance between expressivity and tractability. Note that the latent code consists of μ and σ - the mean and standard deviation of a multivariate Gaussian that may be sampled at test time.

The SVG decoder consists of 4 stacked LSTMs [19] trained with dropout [53, 62, 51]. The final layer is a Mixture Density Network (MDN) [3, 15] that may be stochastically sampled at test time. The LSTM receives as input the previous sampled MDN output, concatenated with the dis-

[2]We have open-sourced tools to reproduce the construction of a subset of the dataset, as well as code to train the proposed model at `https://github.com/tensorflow/magenta`.

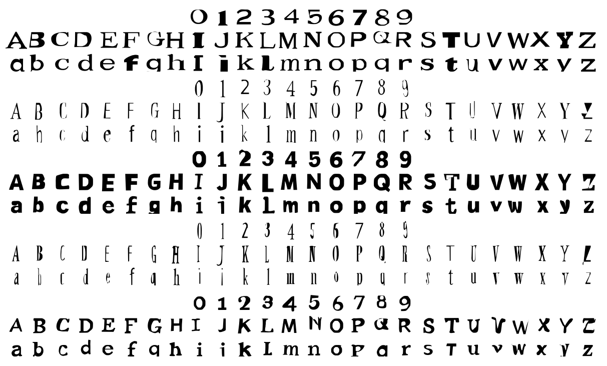


Figure 3: **Selected examples of generated fonts**. Examples generated by sampling a random latent representation z and running the SVG decoder by conditioning on z and all class labels. Each font character is selected as the best of 10 samples. See Figures 10 and 11 in Appendix for additional examples.

crete class label and the latent style representation z. The SVG decoder's loss is composed of a softmax cross-entropy loss over one-hot SVG commands plus the MDN loss applied to the real-valued arguments.

In principle, the model may be trained end-to-end, but we found it simpler to train the two parts of the model separately. The VAE is trained using pixel renderings of the fonts using the Adam optimizer ($\epsilon = 10^{-6}$) [26] for 3 epochs. We employ a high value of β [18], and tune the number of free bits using cross-validation [29]. After convergence, the weights of the VAE are frozen, and the SVG decoder is trained to output the SVG commands from the latent representation z using teacher forcing [61].

Note that both the VAE and MDN are probabilistic models that may be sampled many times during evaluation. The results shown here are the selected best out of 10 samples. Please see Appendix for modeling, training and evaluation details.

Also note that our model is composed of standard architectures. This was an intentional choice to demonstrate that standard methods may provide strong baselines, as we will show in subsequent sections.

4. Results

We compiled a font dataset consisting of 14 M examples. Individual font characters were normalized and converted into SVG format for training and evaluation. We trained a VAE and SVG decoder over 3 epochs of the data and evaluated the results on a hold-out test split. Figures 1 and 3 show selected results from the trained model, but please see Appendix (Figures 10 and 11) for more exhaustive samples highlighting successes and failures. What follows is an analysis of the representational ability of the model to learn and generate SVG specified fonts.

4.1. Learning a smooth, latent representation of font style

We first ask whether the proposed model may learn a latent representation for font style that is perceptually smooth and interpretable. To address this question, we visualize the 32-dimensional font-style z for 1 M examples from the training set and reduce the dimensionality to 2 using UMAP [39]. We discretize this 2D space and visualize the pixel-based decoding of the mean z within each grid location (Figure 4). The purple boxes shows two separate locations of this manifold, where we note the smooth transitions of the characters: (A) represents a non-italics region, while (B) represents an italics one. Further, local directions within these regions also reveal visual semantics: within (A), from left-to-right we note a change in the amount of serif, while top-to-bottom highlights a change in boldness.

Next, we examine whether this smooth space translates into perceptually meaningful decodings of SVGs. We visualize linear interpolations between z for pairs of SVGs from the dataset (Figure 4, 1-6). Note the smooth transition between the decoded SVGs, despite the fact that each SVG decoding is composed of a multitude of different commands. For instance, note that in the top row, each SVG is composed of 15-30 commands even though the perceptual representation appears quite smooth.

4.2. Exploiting the latent representation for style propagation

Because the VAE is conditioned on the class label, we expect that the latent representation z would only encode the font style with minimal class information [28]. We wish to exploit this model structure to perform style propagation across fonts. In particular, we ask whether a *single character* from a font set is sufficient to infer the rest of the font set in a visually plausible manner [47, 20].

To perform this task, we calculate the latent representation z for a single character and condition the SVG decoder on z as well as the label for all other font characters (i.e. `0`-`9`, `a`-`z`, `A`-`Z`). Figure 5 shows the results of this experiment. For each row, z is calculated from the character in the red box. The other characters in that row are generated from the SVG decoder conditioned on z.

We observe a perceptually-similar style consistently within each row [47, 20]. Note that there was no requirement during training that the same point in latent space would correspond to a perceptually similar character across labels – that is, the consistency across class labels was learned in an unsupervised manner [47, 20]. Thus, a single value of z seems to correspond to a perceptually-similar set of characters that resembles a plausible fontset.

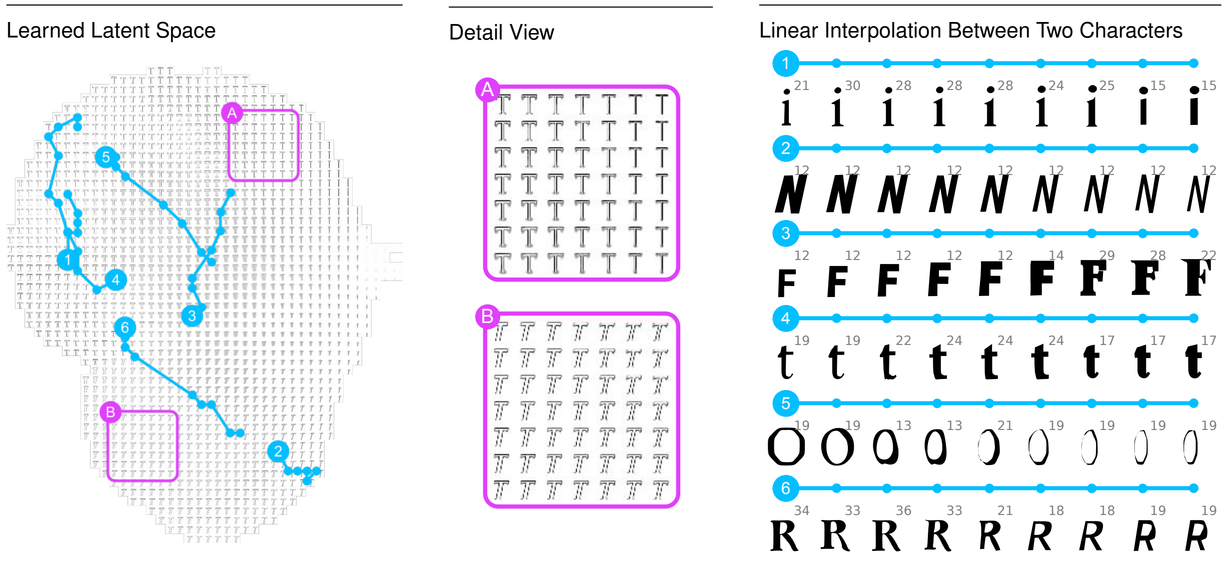


Figure 4: **Learning a smooth, latent representation of font style**. UMAP visualization [39] of the learned latent space z across 1 M examples (left). Purple boxes (A, B) provide a detail view of select regions. Blue lines (1-9) indicate *linear* interpolations in the full latent space z between two characters of the dataset. Points along these linear interpolations are rendered as SVG images. Number in upper-right corner indicates number of strokes in SVG rendering. Best viewed in digital color.

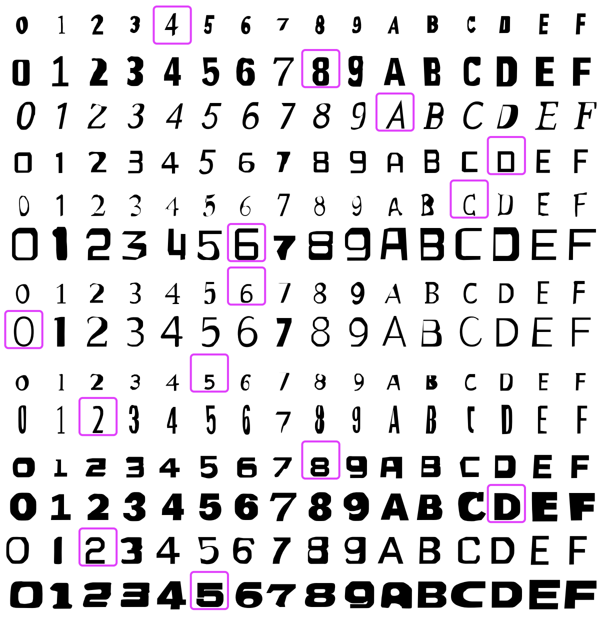


Figure 5: **Exploiting the latent representation for style propagation**. A single character may provide sufficient information for reconstructing the rest of a font set. The latent representation z for a font is computed from a *single* character (purple box) and SVG images are generated for other characters from z.

Additionally, we observe a large amount of style variety across rows (i.e. different z) in Figure 5. The variety indicates that the latent space z is able to learn and capture a large diversity of styles observed in the training set as observed in Figure 4.

Finally, we also note that for a given column the decoded glyph does indeed belong to the class that was supplied to the SVG decoder. These results indicate that z encodes style information consistently across different character labels, and that the proposed model largely disentangles class label from style.

A natural extension to this experiment is to ask if we could systematically improve the quality of style propagation by employing more then a single character. We address this question by calculating the latent representation z for multiple characters and employ the average z for style propagation to a new set of characters (Figure 6). We observe a systematic improvement in both style consistency and quality of individual icon outputs as one conditions on increasing numbers of characters.

To quantify this improvement in style consistency, we render the generated characters and calculate the associated style z for each character. If the method of style propagation were perfectly self-consistent, we would expect that the z across all generated characters would be identical. If, however, the style propagation were not consistent, the inferred z would vary across each of the generated characters. To calculate the observed improvement, we measure the vari-

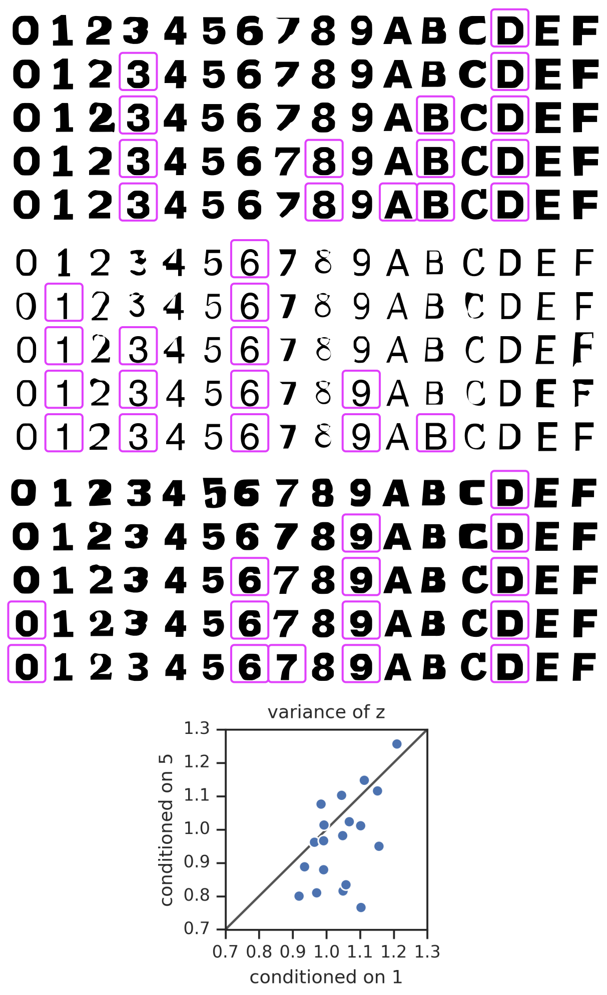


Figure 6: **Conditioning on increasing numbers of characters improves style propagation.** Top: Layout follows Figure 5. The average latent representation z for a font is computed from a set of characters (purple boxes) and SVG images are generated for other characters from z. Note that increasing number of characters (purple boxes) improves the consistency and quality of the style propagation. Bottom: For all generated characters, we calculate a corresponding z and measure the variance of z over all generated characters within a font. Lower variance in z indicates a more visually consistent font style. Each dot corresponds to the observed variance in z when conditioned on 1 or 5 characters. Note that most fonts contain higher consistency (i.e. lower variance) when conditioned on more characters.

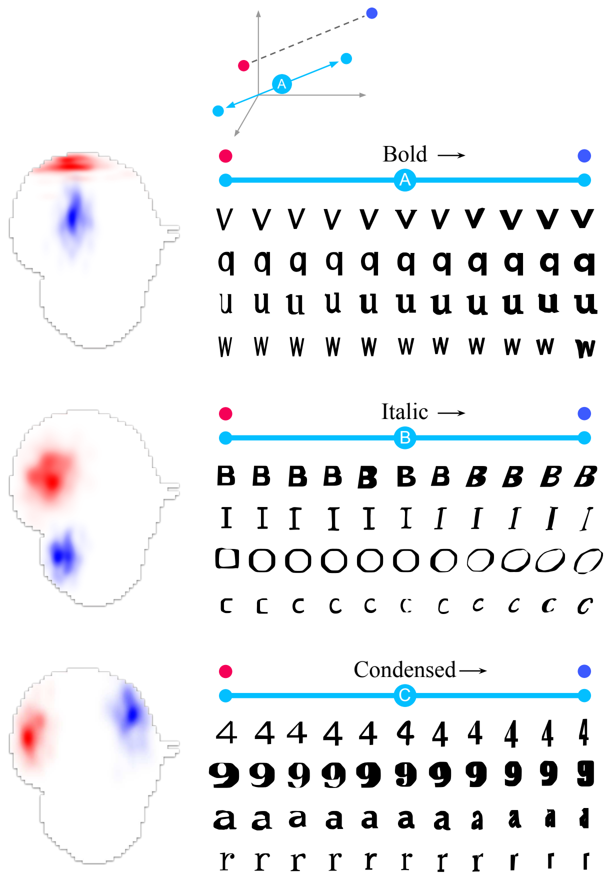


Figure 7: **Building style analogies with the learned representation**. Semantically meaningful directions may be identified for globally altering font attributes. Top row: Bold (blue) and non-bold (red) regions of latent space (left) provide a vector direction that may be added to arbitrary points in latent space (A) for decreasing or increasing the strength of the attribute. Middle and bottom rows: Same for italics (B) and condensed (C).

ance of z across all generated characters for each of 19 fonts explored with this technique when conditioned on on 1 or 5 characters (Figure 6, bottom). Indeed, we observe that conditioning on more characters generally decreases the variance of the generated styles, indicating that this procedure improves style consistency. Taken together, we suspect that these results on style progagation suggest a potential direction for providing iterative feedback to humans for synthesizing new fonts (see Discussion).

4.3. Building style analogies with the learned representation

Given that the latent style is perceptually smooth and aligned across class labels, we next ask if we may find semantically meaningful directions in this latent space. In particular, we ask whether these semantically meaningful directions may permit global manipulations of font style.

Inspired by the work on word vectors [40], we ask whether one may identify analogies for organizing the space of font styles (Figure 7, top). To address this question, we select positive and negative examples for semantic concepts of organizing fonts (e.g. bold, italics, condensed) and identify regions in latent space corresponding to the presence or absence of this concept (Figure 7, left, blue and red, respectively). We compute the average z_{red} and z_{blue}, and define the concept direction $c = z_{blue} - z_{red}$.

We test if these directions are meaningful by taking an example font style z^* from the dataset (Figure 7, right, yellow), and adding (or subtracting) the concept vector c scaled by some parameter α. Finally, we compute the SVG decodings for $z^* + \alpha c$ across a range of α.

Figure 7 (right) shows the resulting fonts. Note that across the three properties examined, we observe a smooth interpolation in the direction of the concept modeled (e.g.: first row v becomes increasingly bold from left to right). We take these results to indicate that one may interpret semantically meaningful directions in the latent space. Additionally, these results indicate that one may find directions in the latent space to globally manipulate font style.

4.4. Quantifying the quality of the learned representations

Almost all of the results presented have been assessed qualitatively. This is largely due to the fact that the quality of the results are assessed based on human judgements of aesthetics. In this section, we attempt to provide some quantitative assessment of the quality of the proposed model.

Figure 8a (top) shows the training dynamics of the model as measured by the overall training objective. Over the course of training 3 epochs, we find that the model does indeed improve in terms of likelihood and plateaus in performance. Furthermore, the resulting model does not overfit on the training set in any significant manner as measured by the log-likelihood.

Figure 8a (bottom) shows the mean negative log likelihoods for the examples in each class of the dataset. There is a small but systematic spread in average likelihood across classes. This is consistent with our qualitative results, where certain classes would consistently yield lower quality SVG decodings than others (e.g. 7 in Figure 5).

We can characterize the situations where the model performs best, and some possible causes for its improved performance. Figure 8b shows the negative log likelihoods of examples from the test set of a given class, as a function of their sequence lengths. With longer sequences, the variance of log likelihoods increase. For the best performing class (8, top) the loss values also trend downwards, whereas for the worst performing (7, bottom), the trend stays relatively level. This means that the model had a harder time reliably learning characters especially with longer sequence lengths.

Finally, in order to see what makes a given character hard or easy to learn, we examine test examples that achieved high and low loss, at different sequence lengths. Figure 8b reveals that for any class characters with high loss are generally highly stylized, regardless of their sequence lengths (red, blue), whereas easier to learn characters are more commonly found styles (yellow, green).

4.5. Limitations of working with a learned, stochastic, sequential representation

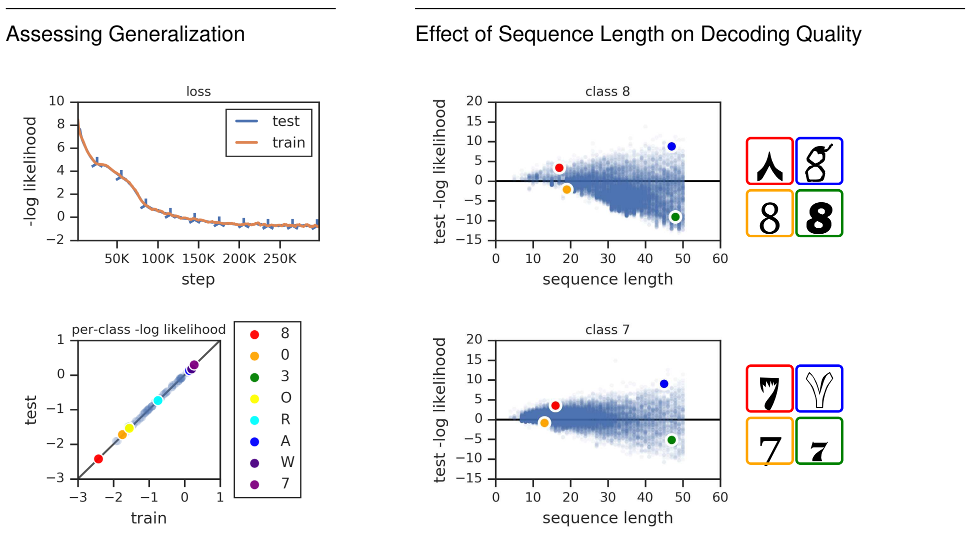
Given the systematic variability in the model performance across class label and sequence length, we next examine how specific features of the modeling choices may lead to these failures. An exhaustive set of examples of model failures is highlighted in in Figure 11 of the Appendix. We discuss below two common modes for failure due to the sequential, stochastic nature of the generative model.

At each stochastic sampling step, the proposed model may select a low likelihood decision. Figure 9 (top left) highlights how a mistake in drawing 3 early on leads to a series of errors that the model was not able to error correct. Likewise, Figure 9 (bottom left) shows disconnected start and end points in drawing 6 caused by the accumulation of errors over time steps. Both of these errors may be remedied through better training schedules that attempt to teach forms of error correction [1], but please see Discussion.

A second systematic limitation is reflected in the uncertainty captured within the model. Namely, the proposed architecture contains some notion of confidence in its own predictions as measured by the variance σ^2 in the VAE latent representation. We visualize the confidence by color-coding the UMAP representation for the latent style z by σ^2 (Figure 9, right). Lighter green colors indicate high model confidence reflected by lower VAE variance. Areas of high confidence show sharp outputs and decode higher quality SVGs (Figure 9 right, pink). Conversely, areas with lower confidence correspond to areas with higher label noise or more stylized characters. These regions of latent space decode lower quality SVGs (Figure 9 right, blue). Addressing these systematic limitations is a modeling challenge for building next generation generative models for vector graphics (see Discussion).

5. Discussion

In the work we presented a generative model for vector graphics. This model has the benefit of providing a scale-invariant representation for imagery whose latent representation may be systematically manipulated and exploited to perform style propagation. We demonstrate these results on a large dataset of fonts and highlight the limitations of a sequential, stochastic model for capturing the statistical dependencies and richness of this dataset. Even in its present form, the current model may be employed as an assistive agent for helping humans design fonts in a more time-efficient manner [6, 47]. For example, a human may



(a) (b)

Figure 8: **Quantifying the quality of the learned representations**. (a) Top: Negative log-likelihood for training and test datasets over 3 epochs. Bottom: Negative log-likelihood for selected, individual classes within the dataset. (b, left) Test negative log-likelihood of all characters with label 8 (top) and 7 (bottom) as a function of the number of SVG commands. (b, right) Examples from 8 and 7 with few commands and high (red) or low loss (orange). Examples with many commands and high (blue) or low loss (green).

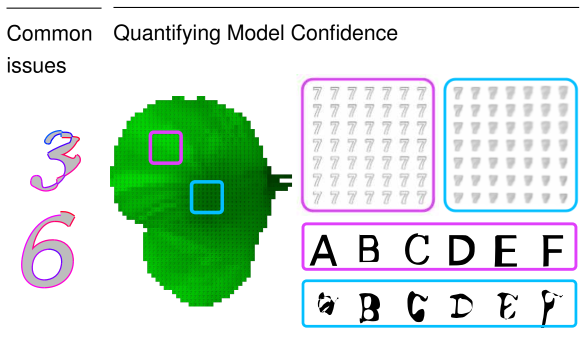


Figure 9: **Limitations of proposed sequential, stochastic generative model.** Left: Low-likelihood samples may result in errors difficult to correct. Color indicates ordering of sequential sample (blue $\rightarrow$ red). Right: Regions of latent space with high variance result in noisy SVG decodings. Latent representation z color coded based on variance: light (dark) green indicates low (high) variance. Visualization of rendered and SVG decoded samples (purple, blue).

design a small set of characters and employ style propagation to synthesize the remaining set of characters (Figure 5, 6).

An immediate question is how to build better-performing models for vector graphics. Immediate opportunities include new attention-based architectures [59] or potentially some form of adversarial training [14]. Improving the model training to afford the opportunity for error correction may provide further gains [1].

A second direction is to employ this model architecture on other SVG vector graphics datasets. Examples include icons datasets [7] or human drawings [50, 16]. Such datasets reveal additional challenges above-and-beyond the fonts explored in this work as the SVG drawings encompass a larger amount of diversity and drawings containing larger numbers of strokes in a sequence. Additionally, employing color, brush stroke and other tools in illustration as a predicted features offers new and interesting directions for increasing the expressivity of the learned models.

Acknowledgements

We'd like to thank the following people: Diederik Kingma, Benjamin Caine, Trevor Gale, Sam Greydanus, Keren Gu, and Colin Raffel for discussions and feedback; Monica Dinculescu and Shan Carter for insights from designers' perspective; Ryan Sepassi for technical help using Tensor2Tensor; Jason Schwarz for technical help using UMAP; Joshua Morton for infrastructure help; Yaroslav Bulatov, Yi Zhang, and Vincent Vanhoucke for help with the dataset; and the Google Brain and the AI Residency teams.

References

[1] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015. 2, 7, 8

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8), 2013. 1

[3] Christopher M Bishop. Mixture density networks. Technical report, Citeseer, 1994. 2, 3, 11

[4] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 1, 2

[5] Neill DF Campbell and Jan Kautz. Learning a manifold of fonts. *ACM Transactions on Graphics (TOG)*, 33(4):91, 2014. 2, 3

[6] Shan Carter and Michael Nielsen. Using artificial intelligence to augment human intelligence. *Distill*, 2017. https://distill.pub/2017/aia. 2, 7

[7] Louis Clouâtre and Marc Demers. Figr: Few-shot image generation with reptile. *arXiv preprint arXiv:1901.02199*, 2019. 2, 8

[8] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016. 2

[9] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. *Proc. of ICLR*, 2, 2017. 3, 11

[10] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *Advances in Neural Information Processing Systems*, pages 6062–6071, 2018. 1, 2

[11] Jon Ferraiolo. *Scalable vector graphics (SVG) 1.0 specification*. 1, 2

[12] Roy Fox, Richard Shin, Sanjay Krishnan, Ken Goldberg, Dawn Song, and Ion Stoica. Parametrized hierarchical procedures for neural programming. *ICLR 2018*, 2018. 2

[13] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*, 2018. 1, 2, 3

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2, 8

[15] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. 2, 3, 11

[16] David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017. 1, 2, 3, 8

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 12

[18] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017. 4

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2, 3

[20] Douglas Hofstadter and Gary McGraw. Letter spirit: An emergent model of the perception and creation of alphabetic style. 1993. 1, 2, 3, 4

[21] Douglas R Hofstadter. *Fluid concepts and creative analogies: Computer models of the fundamental mechanisms of thought.* Basic books, 1995. 1, 2, 3

[22] Varun Jampani, Sebastian Nowozin, Matthew Loper, and Peter V Gehler. The informed sampler: A discriminative approach to bayesian inference in generative computer vision models. *Computer Vision and Image Understanding*, 136:32–44, 2015. 1

[23] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 1

[24] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015. 2

[25] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*, 2018. 1, 2

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4, 12

[27] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10236–10245, 2018. 1, 2

[28] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014. 3, 4

[29] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pages 4743–4751, 2016. 4, 12

[30] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 3, 11

[31] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *Advances in neural information processing systems*, pages 2539–2547, 2015. 1

[32] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 1

[33] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017. 1

[34] Vincent MK Lau. Learning by example for parametric font design. In *ACM SIGGRAPH ASIA 2009 Posters*, page 5. ACM, 2009. 2, 3

[35] Bryan Loh and Tom White. Spacesheets: Interactive latent space exploration through a spreadsheet interface. In *Workshop on Machine Learning for Creativity and Design*, 2018. 3

[36] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *European Conference on Computer Vision*, pages 154–169. Springer, 2014. 1

[37] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016. 1

[38] Vikash K Mansinghka, Tejas D Kulkarni, Yura N Perov, and Josh Tenenbaum. Approximate bayesian image interpretation using generative probabilistic graphics programs. In *Advances in Neural Information Processing Systems*, pages 1520–1528, 2013. 1

[39] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018. 4, 5, 12

[40] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 6

[41] Vinod Nair, Josh Susskind, and Geoffrey E Hinton. Analysis-by-synthesis by learning to invert generative black boxes. In *International Conference on Artificial Neural Networks*, pages 971–981. Springer, 2008. 1

[42] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016. 2

[43] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 3, 11

[44] Pablo Picasso. The bull (le taureau), states i-x, 1946. Museum of Modern Art (MoMA). Mrs. Gilbert W. Chapman Fund. Lithograph. 1

[45] Vinay Uday Prabhu, Sanghyun Han, Dian Ang Yap, Mihail Douhaniaris, Preethi Seshadri, and John Whaley. Fonts-2-handwriting: A seed-augment-train framework for universal digit classification. *arXiv preprint arXiv:1905.08633*, 2019. 3

[46] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015. 2

[47] John A Rehling. *Letter spirit(part two): modeling creativity in a visual domain*. Indiana University, 2001. 2, 3, 4, 7

[48] Daniel Reisberg and Sheri Snavely. Cognition: Exploring the science of the mind. 2010. 1

[49] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014. 11

[50] Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Transactions on Graphics (proceedings of SIGGRAPH)*, 2016. 2, 8

[51] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. *arXiv preprint arXiv:1603.05118*, 2016. 3, 11

[52] Eero P Simoncelli and Bruno A Olshausen. Natural image statistics and neural representation. *Annual review of neuroscience*, 24(1):1193–1216, 2001. 1

[53] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 3, 11

[54] Walt Stanchfield. Gesture drawing for animation. *Washington: Leo Brodie*, 2007. 1

[55] Rapee Suveeranont and Takeo Igarashi. Example-based automatic font generation. In *International Symposium on Smart Graphics*, pages 127–138. Springer, 2010. 2, 3

[56] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016. 2

[57] Aaron van den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315, 2017. 1, 2

[58] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, et al. Tensor2tensor for neural machine translation. *arXiv preprint arXiv:1803.07416*, 2018. 3, 11

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. 8

[60] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 2

[61] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989. 4

[62] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014. 3, 11

[63] Douglas E Zongker, Geraldine Wade, and David H Salesin. Example-based hinting of true type fonts. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 411–416. ACM Press/Addison-Wesley Publishing Co., 2000. 2