Learning a font from a picture

Rob Zinkov and N. Siddharth

1 Introduction and Motivation

Images frequently have some underlying structure we are trying to extract. This could for example be a set objects or some scene description. In this work, we are proposing to extract objects that can be described by some domain-specific language (DSL) and then these objects can be combined somehow to form a description of the original image.

More specifically, we will be detecting and extracting letters from an image which we will then describe using a font language that allows the letter to be drawn in different sizes and orientations.

The font language is inspired by systems like MetaFont[Knuth, 1982]. Fonts in this system describe individual letters as a mathematical forms which may then be drawn as bitmaps of various sizes and positions. Figure 1 shows a β might be rendered as shown in Figure 2. Implicitly assumed in using a font system is that individual letters share stylistic attributes with one another.

We have a more generic model for Omniglot that is end-to-end differentiable. Which is trainable with variational inference while retaining a lot of the structural information. Along with non-parametric bits.

Consider just single characters of Omniglot. Characters from the small alphabet have stylistic similarities and motifs which can be learned and used to learn not just to draw existing letters, but also letters we see only once or not at all.

The contribution of this work is to introduce a system to generate a set of fonts for letters directly from image data. The work has no prior knowledge of what alphabet the letters belong to and only assumes they are non-overlapping in the image.

```
u#:=4/9pt#;
define_pixels(u);
beginchar(66,13u#,16u#,5u#);"Letter beta";
    x1=2u; x2=x3=3u;
    bot y1=-5u; y2=8u; y3=14u;
    x4=6.5u; top y4=h;
    z5=(10u,12u);
    z6=(7.5u,7.5u); z8=z6;
    z7=(4u,7.5u);
    z9=(11.5u,2u);
    z0=(5u,u);
    penpos1(2u,20);
    penpos2(.5u,0);
    penpos3(u,-45);
    penpos4(.8u,-90);
    penpos5(1.5u,-180);
    penpos6(.4u,150);
    penpos7(.4u,0);
    penpos8(.4u,210);
    penpos9(1.5u,-180);
    penpos0(.3u,20);
    pickup pencircle;
    penstroke z1e..z2e..z3e..z4e..z5e..z6e..{up}z7e..z8e..z9e..{up}z0e;
    labels(range 1 thru 9);
endchar;
end
```

Figure 1: Example MetaFont program

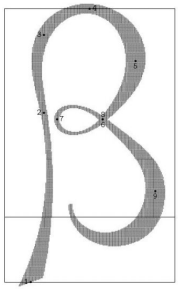

Figure 2: Render of above MetaFont program

2 Background

3 ABC

Many of these drawing simulators don't have a nice tractable likelihood, or would benefit from learning a tractable comparison operator. Use *approximate*

Bayesian computation (ABC) we assume you can generate data from a likelihood function along with the prior distribution, but might not be able to evaluate the density of the likelihood. Now to do better than a rejection sampler we don't directly compare our generated data to our observed data, but instead use a discrepancy function. We then sample from our parameters from the prior distribution, sample from our likelihood function, and if our discrepancy function says the generated and observed data are close enough we accept this sample.

Traditionally, the discrepancy function uses some euclidean distance between domain-specific summary statistics. For our discrepancy function, we explore using the convolution filters from a pretrained variational autoencoder (VAE).

4 Related Work

This work is inspired by the Omniglot challenge Lake et al. [2015] which uses a grammar to describe each character as composed of a series of parts and subparts of a character with subparts coming from a library of primitive character strokes. A sampler is used to how these parts and subparts are combined together to then form the character. I have no idea how inference is done for the BPL models that understand them to be a MCMC sampler with a carefully-chosen proposal distribution.

Another piece of related work is SPIRALGanin et al. [2018], Mellor et al. [2019]. This work defines the drawing task as an adversarial RL task where they jointly learn a discriminator and a generator with a WGAN loss. The discriminator is trained to distinguish images from the data distribution and images rendered from the strokes generated. The strokes are generated using a RL policy that chooses a set of strokes to make given how the rendered image looks as it draws. The agent is then trained using A2C which is a form of Reinforce.

We instead choose to describe all characters are a sequence of B-splines only selecting where each spline starts and ends. Our work most closely follows the Ha and Eck [2017], Chen et al. [2017] which uses a CNN to encode each image into a feature representation which is then fed into an autoregressive model such as a LSTM which outputs stroke data.

5 Model

We represent each stroke as B-spline. B-splines are piece-wise polynomials functions defined to be continuous and differentiable. These strokes are defined in terms of the control points were these piece-wise functions meet $t_1, t_2, \ldots, t_n$ called *knots*. A B-spline with order-n degree polynomials can then be defined recursively as follows.

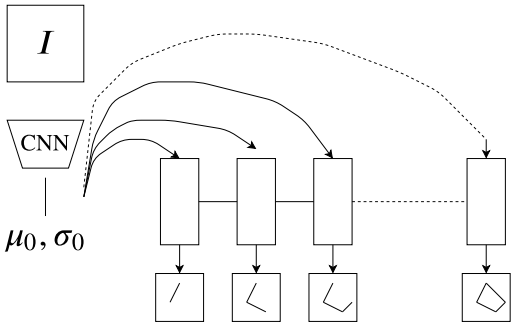

Figure 3: Basic drawing model using a VAE model with a LSTM

$$B_{i,0}(x) := \begin{cases} 1 & \text{if} \quad t_i \leq x < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$B_{i,k}(x) := \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x). \tag{2}$$

Our model extends the work of Chen et al. [2017] where we use a variational autoencoder (VAE). Where a convolutional neural network (CNN) $q_\phi($ is used to encode an image to some feature representation. This feature representation is then fed into a RNN $p_\theta(x \mid y)$ which outputs control points (s_x, s_y) for each part of the spline s_i along with signals for when to start and end each spline.

$$p_\theta(s \mid z) = \prod p_\theta(s_i \mid s_{i-1}, z) \tag{3}$$

$$p_\theta(s_i \mid s_{i-1}, z) = \mathcal{N}((\mu_x, \mu_y), \Sigma) \tag{4}$$

$$\mu_x, \mu_y, \Sigma = \text{LSTM}(s_{i-1}, z) \tag{5}$$

This stroke data is then rendered into an image. Our loss function can then become $\mathbb{E}_{q_\phi(z|z)}[\log p_\theta(x \mid z)]$

While this, model is adequate for learning to draw an image. It's not so good at capturing what makes something a particular letter. We will try to capture the image in a series of glimpses that we try to capture in a single stroke. The style of strokes we settle on will then give us a posterior on styles for an alphabet. We use a Spatial Transformer Network[Jaderberg et al., 2015] to decide on what portion of the image to concentrate on, and then find a B-Spline to draw it.

All the glimpses are then combined and compared with a pixel loss to the original image.

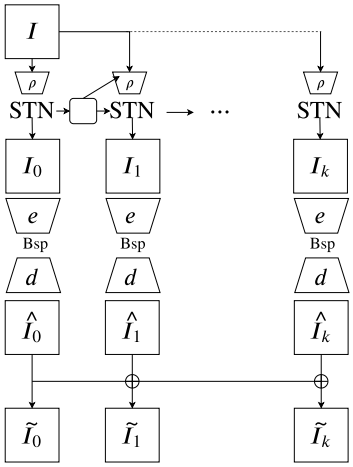

Figure 4: Fancy drawing model using an AIR-like model with multiple glimpses for each stroke

$$p(x,s) = \prod_{i=0}^{K} p(x_i \mid \mu_i, \sigma_i) p(s_i^0 \cdots s_i^l) p(x \mid \tilde{\mu}_i, \tilde{\sigma}_i) \tag{6}$$

$$q(s \mid x) = \prod_{i=0}^{K} q(s \mid x_i) \tag{7}$$

$$= \prod_{i=0}^{K} q(s_i^0 \cdots s_i^l \mid x_i) \tag{8}$$

6 Learning

We train our models in both a supervised manner and a completely unsupervised manner. In a supervised manner, we use datasets of images paired with their stroke data. In an unsupervised manner, we use a VAE model we use only Omniglot images.

7 Experiments

A preliminary experiment was done with MNIST digits where we train a VAE that has two convolutional layers emitting 10 and 20 channels with a kernel size of 5 each. This is followed by two linear layers outputting 50 and then 90 units. These values and then used as control points for a differentiable Bezier curve

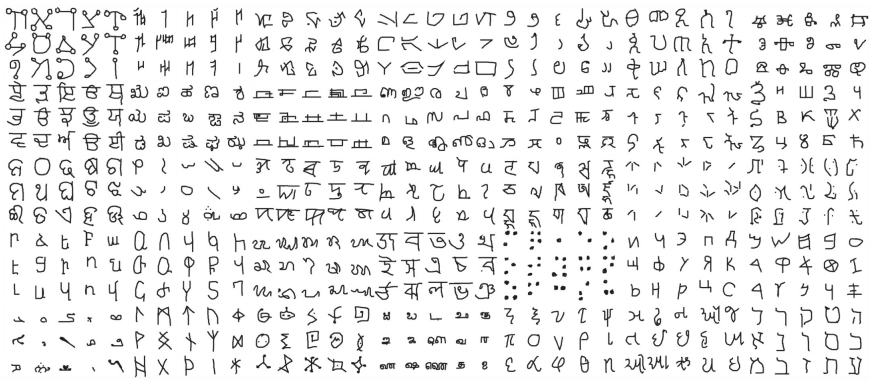

Figure 5: Omniglot characters

renderer.

We use a L1 pixel loss between the rendered image and the true image. That is normalised by the L1 pixel distance between the rendered image and an empty image. This is normalisation is to encourage the control points to be within in the image boundaries.

Additionally, to have a less sparse gradient we use a Gaussian blurred version of the rendered and true image for all our L1 pixel loss comparisons. A neighbourhood of 7 pixels is used for the blur.

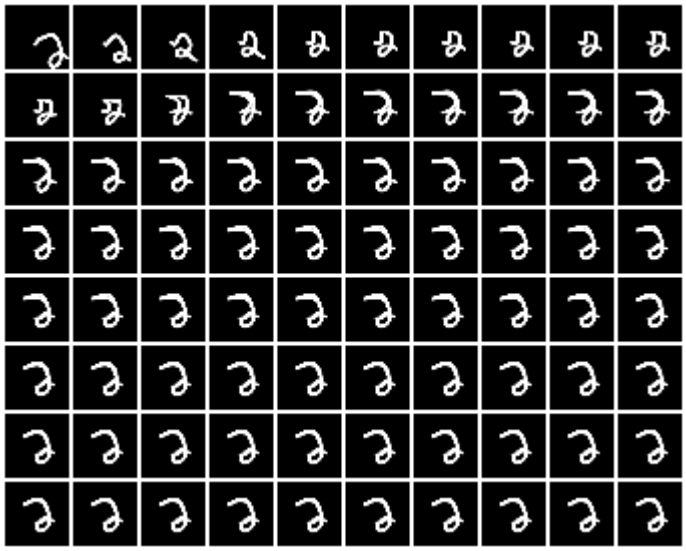

Figure 6: Learning strokes from transposed letter using Adam

Another preliminary experiment involved using Hamiltonian Monte Carlo (HMC) to learn to draw the character. This model uses a Beta(1, 1) distribution for the control points with a Laplace distribution for the likelihood distribution.

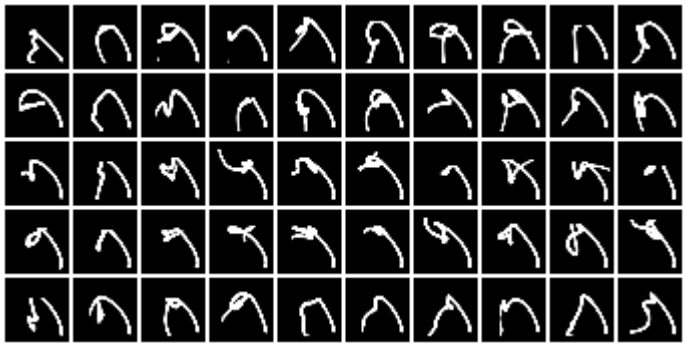

Figure 7: Learning strokes from transposed letter using HMC

Without normalisation training is not stable. This can be seen in the following example where the same initialisation is used but the loss function does not include normalisation against an empty image or gaussian blurring.

Figure 8: Learning strokes from transposed letter

Further we explored if we can learn how to draw a character from random initialisation. Training is still very sensitive to initialisation. The following figures show training from 9 random initialisation for a model with 5, 10, and 20 control points. The character was generated from 20 control points.

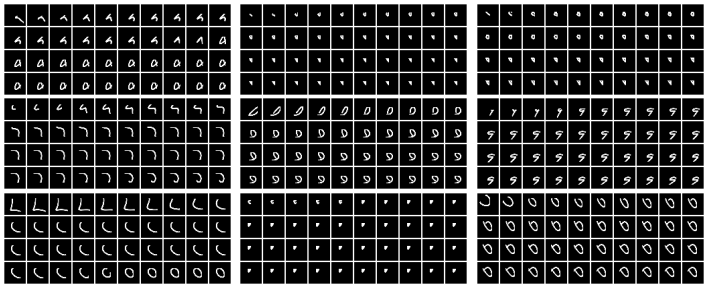

Figure 9: Learning strokes from random initialisation with 5 control points

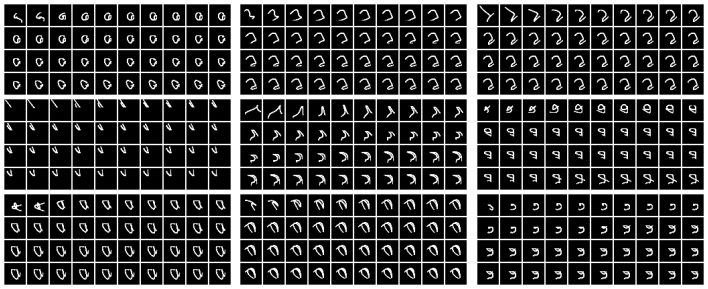

Figure 10: Learning strokes from random initialisation with 10 control points

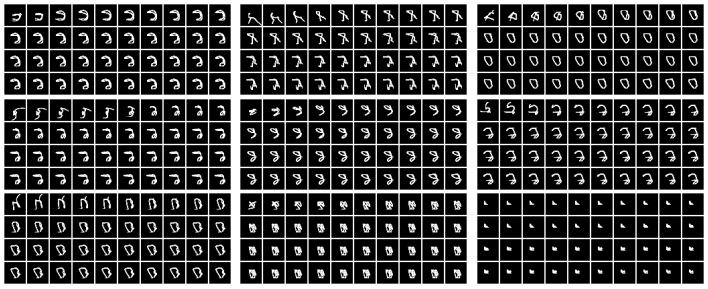

Figure 11: Learning strokes from random initialisation with 20 control points

Another initial experiment will learn to draw single letters from the OmniglotLake et al. [2015] dataset.

This will be followed by experiments using EMNIST[Cohen et al.] and KMNIST[Clanuwat et al., 2018]. The goal being to see if we can draw unseen letters of each alphabet. This shows that we have a high degree of generalisation.

References

Donald E Knuth. The concept of a meta-font. *Visible language*, 16(1):3–27, 1982.

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

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.

John FJ Mellor, Eunbyung Park, Yaroslav Ganin, Igor Babuschkin, Tejas Kulkarni, Dan Rosenbaum, Andy Ballard, Theophane Weber, Oriol Vinyals, and

SM Eslami. Unsupervised doodling and painting with improved spiral. *arXiv preprint arXiv:1910.01007*, 2019.

David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

Yajing Chen, Shikui Tu, Yuqi Yi, and Lei Xu. Sketch-pix2seq: a model to generate sketches of multiple categories. *arXiv preprint arXiv:1709.04121*, 2017.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters (2017). *arXiv preprint arXiv:1702.05373*.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.