UNSUPERVISED LEARNING OF CURVE REPRESENTATION FOR MNIST DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we will describe a general strategy for learning the spline representation of a curve from pixel image unsupervisedly. It contains a trainable encoding network to learn the interior points and moments of the curve, and a determined decoding network to reconstruct the curve image through kernel smoothing. To get neater representations, we introduce length penalty, dense penalty and smooth penalty additional to typical regularization pipeline. Then we show the result of this method in MNIST dataset.

1 INTRODUCTION

Curve fitting is a broadly studied topic with vast application in biomedical imaging. We here focus on learning the centerline from our pixel-wise imaging data. Typical statistical method like principle curve (Goldsmith et al. (2011)), or imaging methods using Bezier curves and B-splines (Cohen et al. (2001)) rely highly on human level assigned initial values or hyper-parameters to get good results, which will then impossible to be processed automatically. Meanwhile auto-encoders (Goodfellow et al. (2016)) are well known efficient unsupervised encoding machines that can potentially learn good representations of curves in the images. But this representation will not be interpretable and we can not get any closed form functional representations of the curve which will be needed in following biomedical research.

Therefore, we want to combine these two sources of methods and get a unsupervised and purly automatic learning machine to get the spline representations of the curves in images.

2 METHODS

A gray scale pixel image is a matrix $\mathbf{A}$ that $\mathbf{A}_{ij}$ represent the image intensity (0-255) at the pixel in i^{th} row and j^{th} column. For simplicity in presenting our method, we normalize every pixel image to be a discrete function F support on $[0,1]^2$ such that

$$F\left(\frac{i}{N}, \frac{j}{N}\right) = \frac{\mathbf{A}_{ij}}{\max_{i,j} \mathbf{A}_{ij}}$$

Therefore $F(x,y) \in [0,1]$.

We are seeking method to learn the functional representation of the centerline $(x(t), y(t))$ in image like:

$$\begin{cases} x(t) = f(t) \\ y(t) = g(t) \end{cases} \tag{1}$$

For $t \in [0,1]$. The method is illustrated as following.

2.1 ENCODING NETWORK

In our method, first we encode the pixel image using a convolutional neural network $\mathcal{C}$, where the last layer is dense layer, to get $4(k+2)$ real values (using sigmoid function to activate just first $2(k+2)$ values to normalize them in $[0,1]$). Here k act like control number in spline method. k is the number of interior points of our cubic spline basis (we also include start and end points, which end in $k+2$). And we hope the encoding net will learn such that the first $2(k+2)$ values (in $[0,1]$) represent the xy coordinates of $k+2$ important points in the centerline of the image, meanwhile the last $2(k+2)$ real values represent the xy derivative of centerline in those important points. To achieve this we need to specific our decoding network.

2.2 Sampling from curve representation

In the decoding network, we will treat the $4(k+2)$ output of $\mathcal{F}$ as they are truly xy coordinates and derivatives and the denote them as $x_1, \cdots, x_{k+2}, y_1, \cdots, y_{k+2}, v_1^x, \cdots, v_{k+2}^x, v_1^y, \cdots, v_{k+2}^y$. Then we do a cubic spline fit, we connect each (x_i, y_i) and (x_{i+1}, y_{i+1}) with a cubic curve. Since we know the xy coordinates and derivatives, we can directly solve all parameters in cubic functions, and get a spline representation $(\hat{x}(t), \hat{y}(t))$. We then sample uniformly T points from the functional representation, and denote them as $(\hat{x}_1, \hat{y}_1), \cdots, (\hat{x}_T, \hat{y}_T)$.

Notice that this step can be done purely by matrix multiplication. If P is the transform from $(x_i, y_i, x_{i+1}, y_{i+1}, v_i^x, v_i^y, v_{i+1}^x, v_{i+1}^y)$ to the parameters of cubic base functions. And B be the evaluation matrix of basis function on uniformly distributed points of $[0,1]$. Then after rearrange $\mathcal{R}$ the output $\mathcal{C}(F)$, we can write that $(\hat{x}_1, \hat{y}_1, \cdots, \hat{x}_T, \hat{y}_T)$ = $BP\mathcal{R}(\mathcal{C}(F))$.

2.3 Reconstruction image from kernel smoothing

After we get the sampled points from the curve, we can reconstruct the pixel image to compare with true image (just input image). To do so, we use a kernel smoothing. Let the reconstructed image (before normalization) F^r be :

$$F^r(u,v) = \sum_{t=1}^{T} \exp\left\{ -\frac{(\hat{x}_t - u)^2 + (\hat{y}_t - v)^2}{2\sigma^2} \right\} \qquad (u,v) \in [0,1]^2 \tag{2}$$

And we then need to normalize F^r to be in $[0,1]$. This can be done by multiple ways, say here we try the reconstructed $\hat{F} = \tanh^2(F^r)$ to act like an unlinear exposure. For this whole reconstruction process from sampled points, we denote it as $\mathcal{K}$.

After combining all these processes, we have that $\hat{F} = \mathcal{K}(BP\mathcal{R}(\mathcal{C}(F)))$. Notice that every functional here is differentiable therefore parameters in $\mathcal{K}$ and $\mathcal{C}$ can be trained using typical SGD algorithm if we specify the loss.

2.4 Losses

Using the same notation as above, we introduce several needed loss for the training step. $\hat{F} = \mathcal{K}_\sigma(BP\mathcal{R}(\mathcal{C}_\theta(F)))$

1. L^2 **Distance**:
   The main part loss is just the difference between reconstructed image and true image, which is $||\hat{F} - F||_F^2$.
2. L^2 **Regularization Penalty**:
   To avoid over-fitting in encoding stage, we need to penalize θ the complexity of the encoding net, which introduce loss part as $||\theta||_F^2$.
3. **Length Penalty**:
   To avoid over-fitting in the reconstruction stage, we need to penalize the length of the fitted curve. If we don't do that, the algorithm tend to use thin curve and go back and back again through the white part of the image to fill it, which is not neat in representation and will increase the variation.
   The length penalty is approximated from sampled points.

$$LP_\theta = \sum_{t=2}^{T} \sqrt{(\hat{x}_t - \hat{x}_{t-1})^2 + (\hat{y}_t - \hat{y}_{t-1})^2} \tag{3}$$

For multiple images, it will be the mean of quantity 3.

4. **Smooth Penalty**: To assist avoiding over-fitting in reconstruction stage, we also want the curve to be smooth enough. Therefore the derivative should not change a lot. The smooth penalty should be the mean of:

$$SP_\theta = \sum_{i=2}^{k+2} (v_i^x - v_{i-1}^x)^2 + (v_i^y - v_{i-1}^y)^2 \tag{4}$$

5. **Dense Penalty**: We also want the interior points learned by the encoding net be somehow separate from each other and therefore may be more representative. The dense penalty will be the mean of:

$$DP_\theta = -\sum_{i=2}^{k+2} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{5}$$

Therefore we aim to train the model by solving:

$$\min_{\theta,\sigma} ||F - \mathcal{K}_\sigma(BP\mathcal{R}(\mathcal{C}_\theta(F)))||^2 + \lambda||\theta||^2 + \gamma_l LP_\theta + \gamma_d DP_\theta + \gamma_s SP_\theta \tag{6}$$

And once we get the solution, the curve spline representation have already been derived from sampling points stage.

3 Theory

Theorem 3.1. *If image dimension $N \to \infty$, true $\sigma \to 0$ and encoding network $\mathcal{C}$ complex enough, we have that under solution of program*

$$\min_{\theta,\sigma} ||F - \mathcal{K}_\sigma(BP\mathcal{R}(\mathcal{C}_\theta(F)))||^2 \tag{7}$$

$\mathcal{C}_\theta(F)$ truly represent $k+2$ points and corresponding derivatives in the underlying curve.

Proof. The proof is done by contradiction. Suppose $\mathcal{C}$ is complex enough to capture the true relationship from image to interior points and derivatives (since neural network can approximate any function, this is achievable when $\mathcal{C}$ is complex enough). Then just show that moving one point to nearest point in underlying curve or correcting one derivative of points in the curve will always make program 7 smaller. Therefore the output are truly on the curve. □

Theorem 3.2. *Under the same condition of theorem 3.1 and suppose the true underlying curve is diffeomorphic with a closed interval in $\mathbb{R}^2$. Then for $\gamma_l \to 0$, under the solution of program:*

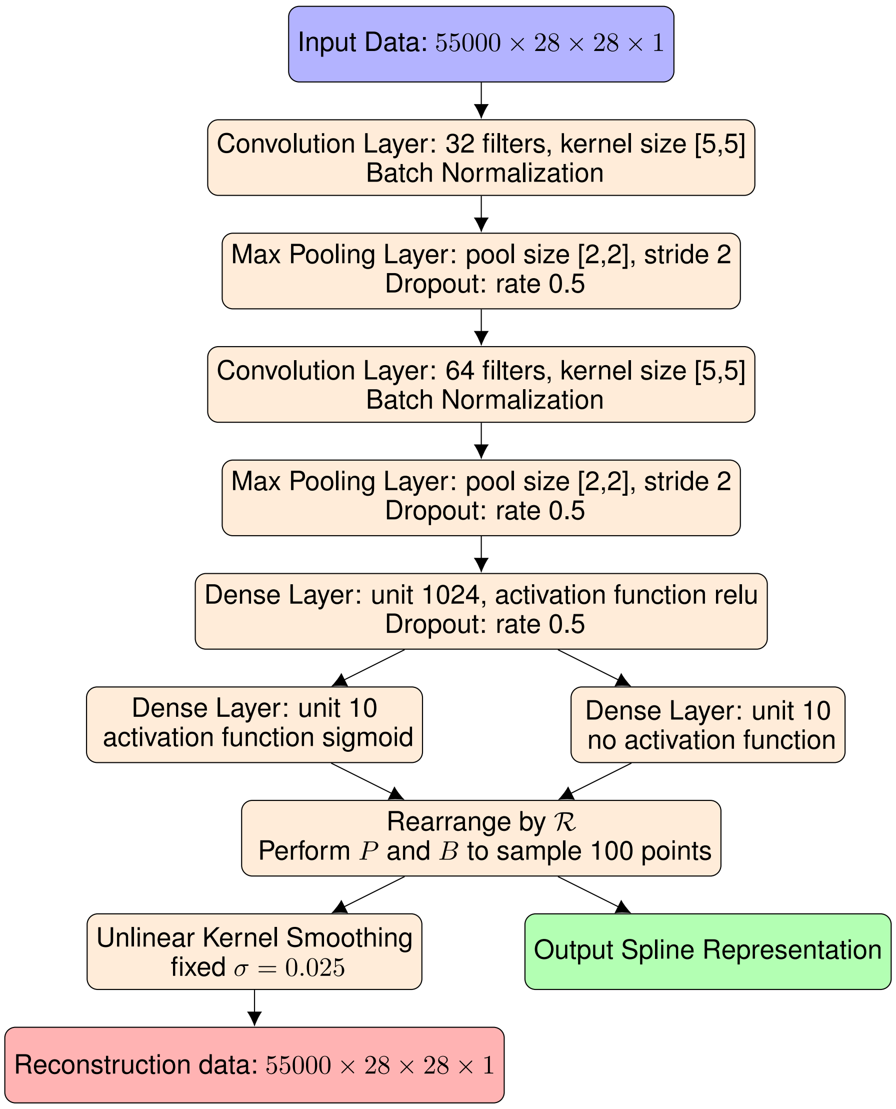
$$\min_{\theta,\sigma} ||F - \mathcal{K}_\sigma(BP\mathcal{R}(\mathcal{C}_\theta(F)))||^2 + \gamma_l LP_\theta \tag{8}$$

there exists an orientation of the curve such that the points of $\mathcal{C}_\theta(F)$ are in the same order and the first/last point in $k+2$ is the start/end point of the curve.

Proof. The proof is done by contradiction. For the output of $\mathcal{C}$, show that reordering the output points in order and move first and last point to start and end point will decrease the value of programming 8. □

4 Experiment

In the experiment we used MNIST hand-written digit dataset. The illustration of our model is following.


Input Data: $55000 \times 28 \times 28 \times 1$
Convolution Layer: 32 filters, kernel size [5,5]
Batch Normalization
Max Pooling Layer: pool size [2,2], stride 2
Dropout: rate 0.5
Convolution Layer: 64 filters, kernel size [5,5]
Batch Normalization
Max Pooling Layer: pool size [2,2], stride 2
Dropout: rate 0.5
Dense Layer: unit 1024, activation function relu
Dropout: rate 0.5
Dense Layer: unit 10
activation function sigmoid
Dense Layer: unit 10
no activation function
Rearrange by $\mathcal{R}$
Perform P and B to sample 100 points
Unlinear Kernel Smoothing
fixed $\sigma = 0.025$
Output Spline Representation
Reconstruction data: $55000 \times 28 \times 28 \times 1$

The combination of hyper-parameters used in the experiment are $k = 3$, $\lambda = 0.001$, $\gamma_l = 0.005$, $\gamma_d = 0.001$, $\gamma_s = 0.01$. Typically we don't need to tune all γs separately. Let $\gamma_l = 5\gamma$, $\gamma_d = \gamma$, $\gamma_s = 10\gamma$ will be sufficient.

Then the model is trained by an AdamOptimizer with learning rate 0.005 and batch size 64. And it will converge well after 4 epoch. See the figure 1.

During the training process, we output compare reconstruction image vs original and also plot fitted curve in validation set. See the figure 2a, 2b, 3a, 3b, 4a, 4b. Notice that in those plots.

- First line is true digit pictures from validation set.
- Second line is the reconstruction images from the curve fitting model.
- Third line is curve plot, there are also 5 points which indicate the output of $\mathcal{C}(F)$ and the order is from big size point to small size point.

We can see that the result is quite good but still not perfect because it didn't output the most neat representation. Also this trained model can not generalize well to handwritten letter data because it just learned digit structure. See the behavior in figure 5

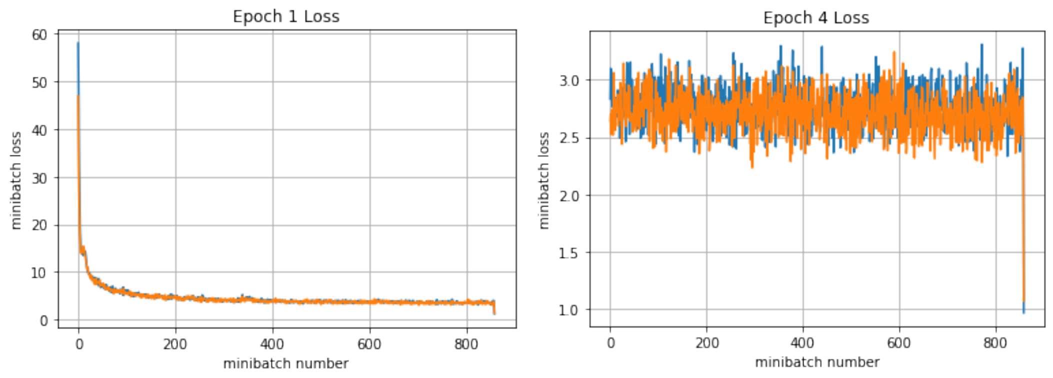
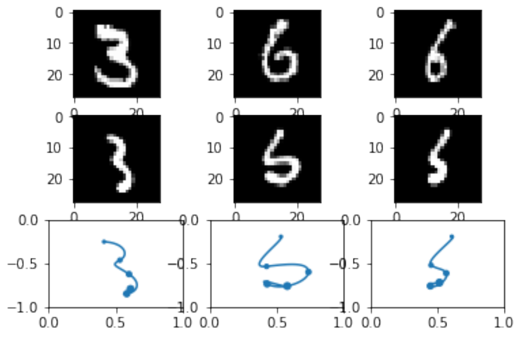
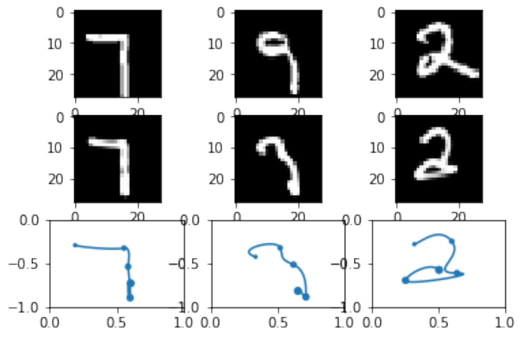


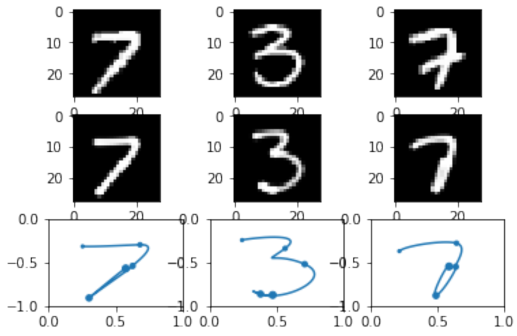
Figure 1: Convergence of loss function



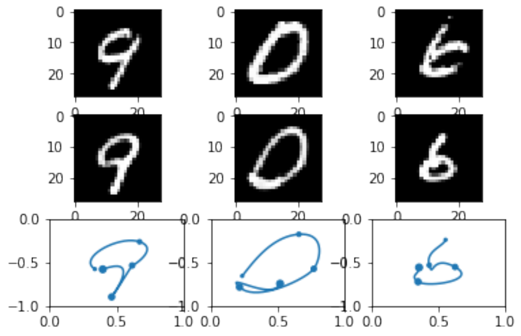
(a) Starting Behavior



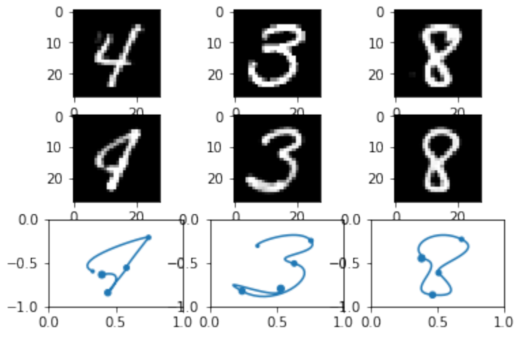
(b) Some struggling digits



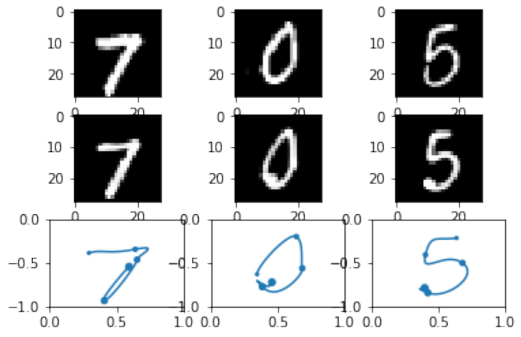
(a) Robust Behavior



(b) Good performance



(a) Good Performance 2



(b) Good Performance 3

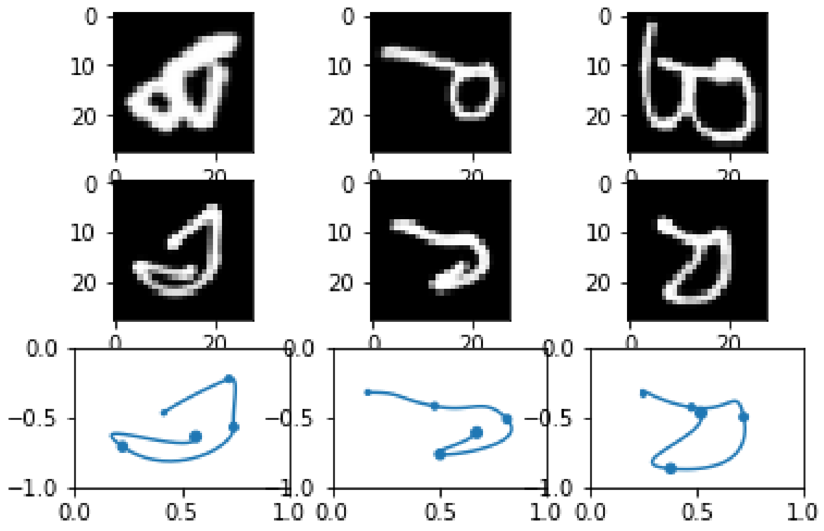


Figure 5: Can not generalize to shape it haven't seen before

References

Elaine Cohen, Richard F Riesenfeld, and Gershon Elber. *Geometric modeling with splines: an introduction*. AK Peters/CRC Press, 2001.

Jeff Goldsmith, Brian Caffo, Ciprian Crainiceanu, Daniel Reich, Yong Du, and Craig Hendrix. Nonlinear tube-fitting for the analysis of anatomical and functional structures. *The annals of applied statistics*, 5(1):337, 2011.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.