LONG RANGE CONSTRAINTS FOR NEURAL TEXTURE SYNTHESIS USING SLICED WASSERSTEIN LOSS

Liping Yin

Michigan State University
Department of Mathematics and CMSE
619 Red Cedar Rd, East Lansing, MI

Albert Chua

Michigan State University
Department of Mathematics
619 Red Cedar Rd, East Lansing, MI

ABSTRACT

In the past decade, exemplar-based texture synthesis algorithms have seen strong gains in performance by matching statistics of deep convolutional neural networks. However, these algorithms require regularization terms or user-added spatial tags to capture long range constraints in images. Having access to a user-added spatial tag for all situations is not always feasible, and regularization terms can be difficult to tune. It would be ideal to create an algorithm that does not have any of the aforementioned drawbacks. Thus, we propose a new set of statistics for exemplar based texture synthesis based on Sliced Wasserstein Loss and create a multi-scale algorithm to synthesize textures without a user-added spatial tag. Lastly, we study the ability of our proposed algorithm to capture long range constraints in images and compare our results to other exemplar-based neural texture synthesis algorithms.

Index Terms— Neural Texture Synthesis, Sliced Wasserstein Distance, Convolutional Neural Networks

1. INTRODUCTION

1.1. Background on exemplar-based texture synthesis

Texture synthesis has been of interest to researchers for over half a century starting with [1]. The goal is to take a reference texture and use statistical information about the reference texture to generate a new, different texture with similar properties as the reference texture. Until recently, many methods, like [3, 4], have used statistical information of wavelet coefficients for texture generation. However, these methods failed to produce believable synthesis for images with complex structures, which suggested more statistical information was needed to capture the essence of a texture.

Neural texture synthesis, which involves the use of deep neural networks to generate textures, started with the work of Gatys et. al. in [6, 7]. The authors used the mean squared error between gram matrices of feature maps of VGG19 as a loss function for texture synthesis. While the results in [7] were very strong compared to previous work at the time of publication, the algorithm failed to capture long range constraints of images, like the alignment of a wall. Numerous papers have proposed methods of imposing long range constraints to synthesized images such as [8, 9, 10, 11]. These approaches all use the method in [7] and add some type of multi-scale procedure or add multiple regularization terms, like a fourier spectrum constraint or autocorrelation constraint.

Each of these methods allow for more faithful texture synthesis, but they also have their drawbacks. The method of [9] is the simplest, and does capture more long range constraints. However, the method has a tendency to fail on images with rigid structure, like brick walls. On the other hand, methods like [8, 10, 11] are effective at generating a diverse set of textures, but these methods can be difficult to hyperparameter tune because of the regularization parameters.

Other methods include [12], which uses a generative adversarial network for synthesis. However, this method is much slower than other methods we have described above. Uuniversal texture synthesis [17, 18, 19] trains a network to generate textures in a feedforward fashion, but this is outside the scope of our paper.

1.2. Background on Sliced Wasserstein Loss

Suppose that layer ℓ of an L layer convolutional neural network has N_ℓ channels and M_ℓ pixels in each channel. We denote the feature vector located at pixel m as $F_m^\ell \in \mathbb{R}^{N_\ell}$.

The authors of [15] propose using a different set of statistics instead of the mean squared error between gram matrices. In a manner similar to [3, 14], the authors match the distributions between feature maps, but these feature maps used in [15] are feature maps of VGG19 [5].

With respect a network architecture, let p^ℓ and $\hat{p}^\ell$ be the probability density functions associated with the set of feature vectors $\{F_m^\ell\}$ and $\{\hat{F}_m^\ell\}$. Since our network is discrete, we assume that the probability density functions are always an average of Dirac delta distributions of the form

$$p^\ell(x) = \frac{1}{M_\ell} \sum_{m=1}^{M_\ell} \delta_{F_m^\ell}(x). \tag{1}$$

Let $V \in \mathbb{S}^{N_\ell}$ be a set of random directions on the unit sphere of dimension N_ℓ. For the purpose of this paper, the Sliced Wasserstein Distance between two distributions of features is of the form

$$\mathcal{L}_{\text{SW},\ell}(p^\ell, \hat{p}^\ell) = \mathbb{E}_V[\mathcal{L}_{\text{SW1D}}(p^\ell_V, \hat{p}^\ell_V)], \tag{2}$$

where

$$p^\ell_V := \{\langle F^\ell_m, V\rangle\} \tag{3}$$

is a set consisting of batched projections of the feature maps F^ℓ_m onto the directions V; if we make a vector P^ℓ_V consisting of the elements of p^ℓ_V, the $1D$ Sliced Wasserstein Loss is the 2-norm between sets of sorted projections:

$$\mathcal{L}_{\text{SW1D}}(p^\ell_V, \hat{p}^\ell_V) = \frac{1}{\text{len}(P^\ell_V)} \left\| \text{sort}(P^\ell_V) - \text{sort}(\hat{P}^\ell_V) \right\|_2^2 \tag{4}$$

and the full Sliced Wasserstein loss over all the layers is

$$\mathcal{L}_{\text{SW}}(I_1, I_2) = \sum_{\ell=1}^{L} \mathcal{L}_{\text{SW},\ell}(p^\ell_{V,I_1}, p^\ell_{V,I_2}), \tag{5}$$

for images I_1 and I_2, respectively. For practical applications, one usually will use a loss of the form

$$\mathcal{L}_{\text{SW}}(I_1, I_2) = \sum_{\ell=1}^{L} w_i \mathcal{L}_{\text{SW},\ell}(p^\ell_{V,I_1}, p^\ell_{V,I_2}), \tag{6}$$

where w_i are weight terms that set to zero for layers that are not used. We will use this formulation for the rest of the paper.

2. TEXTURE SYNTHESIS ALGORITHM

2.1. Adding statistics

Instead of just matching distributions via slicing over the channel dimension of the feature maps, we purpose matching more statistics in a very simple way. Consider a set of feature maps $F^\ell \in \mathbb{R}^{H_\ell \times W_\ell \times N_\ell}$. In the original algorithm, we unravel each $H_\ell \times W_\ell$ feature map, consider each pixel m to get feature vectors of length N_ℓ, and project onto direction V in Eq. (5). Another way to reshape the feature maps is to unravel them into H_ℓ different $W_\ell \times N_\ell$ feature vectors, F^ℓ_H (with $F^\ell_{H,n}$ being a vector of all n^{th} pixels of each feature vector), and project them onto directions $V_{H_\ell} \in \mathbb{S}^{H_\ell}$. Analogous to Eq. (3), for the distribution p^ℓ_H associated to feature vectors $\{F^\ell_{H,n}\}$ we can define another set of batched projections given by

$$p^\ell_{V_{H_\ell}} = \{\langle F^\ell_{H,n}, V_{H_\ell}\rangle\}. \tag{7}$$

The corresponding new loss term is

$$\mathcal{L}_{\text{SW},H}(I_1, I_2) = \sum_{\ell=1}^{L} w_i \mathcal{L}_{\text{SW},\ell}\left(p^\ell_{V_{H_\ell},I_1}, p^\ell_{V_{H_\ell},I_2}\right). \tag{8}$$

Intuitively, this loss term accounts for alignment in an image by slicing over the dimension for the height of the feature maps rather than the dimension for the channel of the feature maps. The new loss function we consider is

$$\mathcal{L}_{\text{Slicing}}(I_1, I_2) = \mathcal{L}_{\text{SW}}(I_1, I_2) + \mathcal{L}_{\text{SW},H}(I_1, I_2), \tag{9}$$

which is the sum of Eq. (5) and Eq. (8).

2.2. Algorithm details

For our algorithm, we start with a reference image I_{ref} and a white noise I_{WN}, run for M epochs and we will denote feature map extraction from VGG19 as Extract(I). In our next algorithm, we will assume that we want to synthesize an image the same size as our reference image without any loss of generality.

Algorithm 1: Synthesis Algorithm

1: Set I_{WN} as variable to be updated by optimizer.
2: **for** $k = 1, \ldots, M$ **do**
3: Calculate Extract(I_{WN}).
4: Calculate Extract(I_{ref}).
5: Calculate $\mathcal{L}_{\text{Slicing}}(I_{\text{WN}}, I_{\text{ref}})$.
6: Backpropogate and update I_{WN}.
7: **end for**
8: Return updated I_{WN} as synthesized texture.

The settings for experimentation in Sec. 3 are to use the layers the first 12 layers of VGG19 for slicing over the feature maps and the first two convolutions (after the ReLU) in each convolution block for the other loss term. We also use the L-BFGS optimizer [2] with a learning rate of $\eta = 1$.

A natural question one could ask is if one could try a similar approach for gram matrices. However, the computational cost for this is infeasible based on our experiments so far. Our run times were over an hour for a single scale on small images, even while using a graphics processing unit.

2.3. Multi-scale algorithm

The multi-scale procedure we use below has been used in [11, 13, 14], but not for neural texture synthesis via Sliced Wasserstein Distances. For the multi-scale algorithm at N scales, let $I_{\text{ref},i}$ be the reference image downsampled by a scale factor of 2^i with $i = 0, \ldots, N$, and define the upsampling operator as Upsample(I). Lastly, define the output of Algorithm 1 using the notation $I_{\text{Synthesis}} = \text{SWSynthesis}(I_{\text{input}}, I_{\text{ref}})$, where I_{input} is the input to be optimized via backpropogation, I_{ref} is the reference texture, and $I_{\text{Synthesis}}$ is the output after synthesis.

Algorithm 2: Multi-scale Synthesis Algorithm

1: Initialize $I_{\text{Synthesis}}$ as a white noise that is the same size as the reference texture downsampled by 2^K.
2: **for** $i = 0, \ldots, N$ **do**
3: $\quad I_{\text{Synthesis}} \leftarrow \text{SWSynthesis}(I_{\text{Synthesis}}, I_{\text{ref},K-i})$.
4: $\quad I_{\text{Synthesis}} \leftarrow \text{Upsample}(I_{\text{Synthesis}})$.
5: **end for**
6: Return $I_{\text{Synthesis}}$ as the synthesized texture.

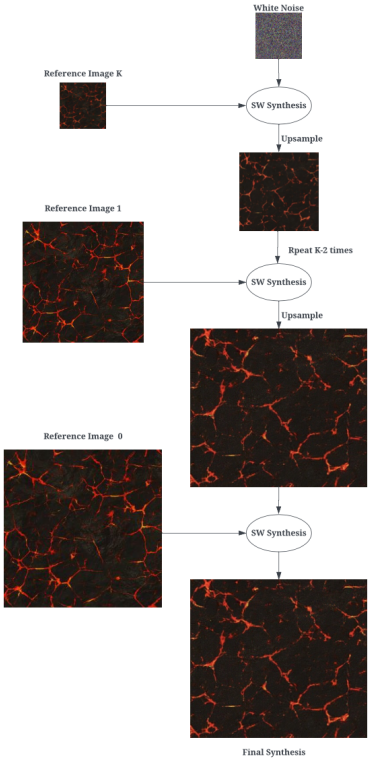

Fig. 1. Multi-scale Texture Synthesis Algorithm Visualization

3. EXPERIMENTS

All textures are taken from the sources in the footnotes[1][2].

3.1. Multi-scale approach without our loss term

We start by evaluating the effectiveness of our new loss term Eq. (8) by trying to use a multi-scale approach with the algorithm in [15] without a spatial tag in Fig. 2. From our results, we see that using a multi-scale approach by itself is not enough to fully capture nonstationary statistics or enforce long range constraints.

[1]https://github.com/omrysendik/DCor/tree/master/Data
[2]https://www.robots.ox.ac.uk/ vgg/data/dtd/

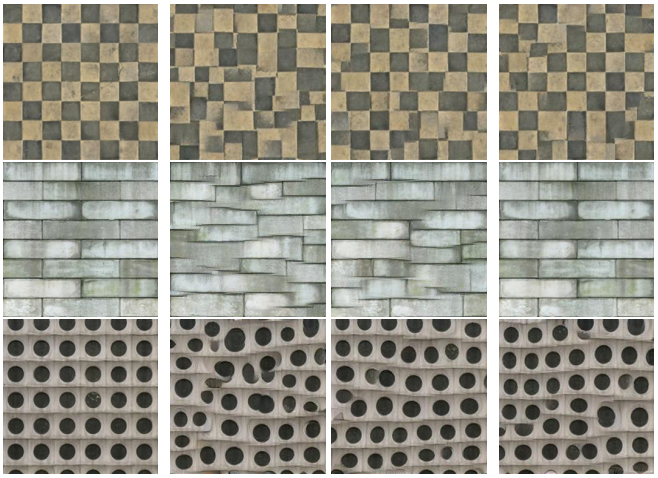

Fig. 2. Synthesis results without the loss term. **Left:** Reference Texture. **Middle Left:** $N = 0$ (the original algorithm without spatial tag). **Middle Right:** $N = 1$. **Right:** $N = 2$.

3.2. Effect of the number of scales

Now we test the effect of the number of scales used in our multi-scale algorithm with the loss term using some textures above and also with some nonstationary textures. The results are in Fig. 3. What we see is that adding more scales helps with alignment and color preservation to some extent. However, when $N = 2$, our second and third textures are repetitions of the original texture. This suggests that $N = 1$ is the best choice for high synthesis quality without repetition.

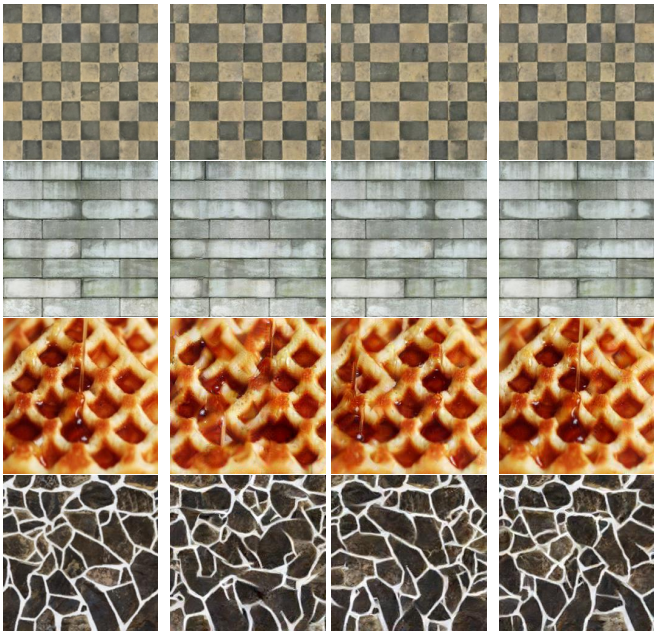

Fig. 3. Synthesis results with our loss term. **Left:** Reference Texture. **Middle Left:** $N = 0$. **Middle Right:** $N = 1$. **Right:** $N = 2$.

3.3. Comparison to other algorithms

We now provide more examples where we compare our results to other authors in Fig. 4 with some parametric and non-parametric textures. Namely, we compare our results of using the code from [15] and [11]. We choose to not include results from [8, 9] because the results of [11] are stronger; additionally, we do not include results from [10] because tests from [10, 11] indicate that the algorithm does not necessarily improve performance for textures containing long range constraints that are not nearly periodic. Our algorithm is run with $N = 1$ and 20 iterations of slicing for each step of Algorithm 1. While our results are not exhaustive, they provide some evidence for us to say our algorithm has the ability to capture long range constraints that other algorithms are not always able to. In all cases, the alignment captured by our synthesis is equal to or better than [15] without the spatial tag. Additionally, our algorithm performs better than [11] on the last two textures because we do not create a repetition like their algorithm or create a result that is too similar to the reference texture.

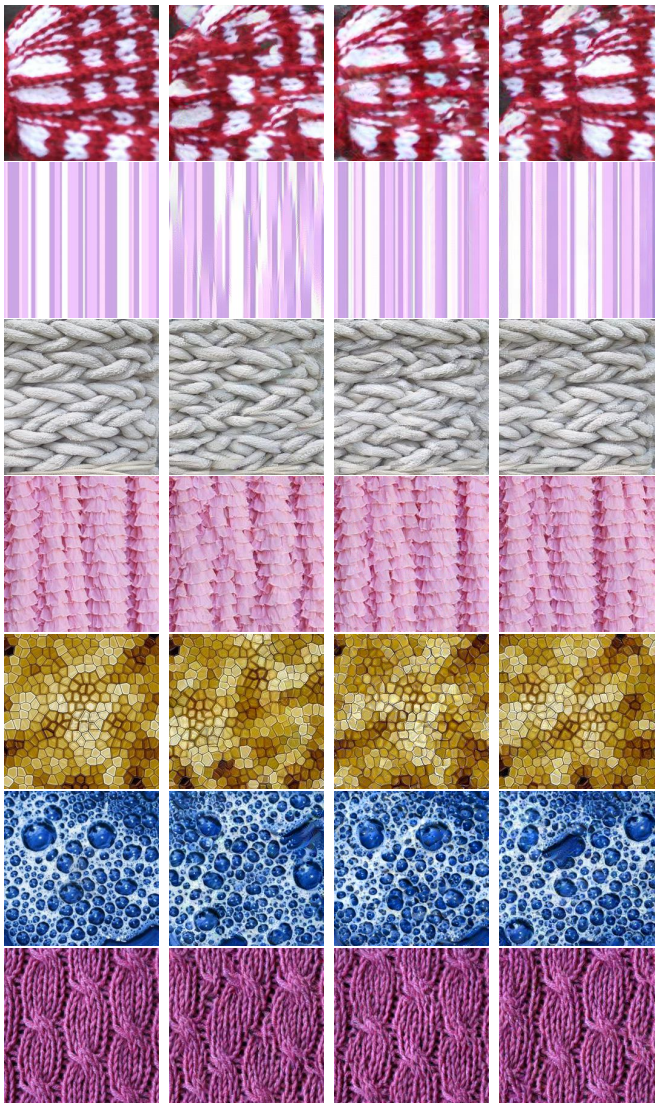

Fig. 4. Synthesis results compared to other models. **Left:** Reference Texture. **Middle Left:** Results from Heitz et. al. [15] without the spatial tag. **Middle Right:** Results from Gonthier et. al. [11]. **Right:** Results from our method.

3.4. Why not three directions?

A natural question to ask is: why not slice over the dimension for the width of the feature maps as well? This would add more statistics to our network and possibly create better synthesis. In the experiments using $K = 0$, we found that this set of statistics leads to reproduction or near reproduction of the reference texture for periodic textures.

3.5. Conclusions

We present a modification of texture synthesis via Sliced Wasserstein Loss that has the ability to add long range constraints without user-added spatial tags. Our extra loss term can be thought of as a regularization term, but unlike traditional regularization terms, like a spectrum constraint, we did not need to add hyperparameter λ to ensure that our new loss converged. This presents multiple benefits over traditional regularization terms.

One interesting question that arises from using Sliced Wasserstein Loss is the following: what effect does network architecture have on synthesis? Texture synthesis using gram matrices of wavelet coefficients with a nonlinearity applied has been done by [16], and the results are very convincing. However, the work in [3] is very similar to [15], just with a wavelet transform instead of a neural network, but the results are poor. This suggests that the depth of a network is very important for synthesis using Sliced Wasserstein Loss. Can deeper networks lead to more faithful synthesis? Or can we get similar results with shallower networks?

4. REFERENCES

[1] Bela Julesz. "Visual pattern discrimination," in *IRE Transactions on Information Theory, vol. 8, no. 2, pp. 84–92, 196*, 1962.

[2] Dong C. Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization," in *Mathematical Programming, vol. 45, pp. 503–528*, 1989.

[3] D. J. Heeger and J. R. Bergen. "Pyramid-based texture analysis/synthesis," in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, ACM*, 1995, pp. 229–238.

[4] Javier Portilla and Eero P Simoncelli. "A parametric texture model based on joint statistics of complex wavelet coefficients," in *International Journal of Computer Vision, vol. 40, no. 1, pp. 49–70*, 2000.

[5] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations (ICLR)*, May 2015.

[6] Leon Gatys, Alexander S Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[7] Leon Gatys, Alexander S Ecker, and Matthias Bethge. "Texture synthesis using convolutional neural networks," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 262–270.

[8] Yann Gousseau, Gang Liu, and Gui-Song Xia. "Texture synthesis through convolutional neural networks and spectrum constraints," in *International Conference on Pattern Recognition (ICPR)*, December 2016.

[9] Xavier Snelgrove. "High-resolution multi-scale neural texture synthesis," in *SIGGRAPH Asia 2017 Technical Briefs, SA '17, New York, NY, USA, 2017. Association for Computing Machinery.*

[10] Omry Sendik and Daniel Cohen-Or. "Deep correlations for texture synthesis," in *ACM Trans. Graph., 36(4)*, July 2017.

[11] Yann Gousseau, Nicolas Gonthier, and Saïd Ladjal. "Texture synthesis through convolutional neural networks and spectrum constraints," in *Journal of Mathematical Imaging and Vision 64:478–492*, 2022.

[12] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. " Non-stationary texture synthesis by adversarial expansion," in *ACM Transactions on Graphics (ToG), vol. 37, ACM, 2018, pp.1-13*, 2018.

[13] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. 'Wasserstein loss for image synthesis and restoration," in *ACM Transactions on Graphics (ToG), vol. 24, ACM, 2005, pp. 795–802*.

[14] Guillaume Tartavel, Gabriel Peyre, and Yann Gousseau, "Variational Texture Synthesis with Sparsity and Spectrum Constraints," in *Journal of Mathematical Imaging and Vision, 52:124–144*, 2015.

[15] Eric Heitz, Kenneth Vanhoey, Thomas Chambon, and Laurent Belcour. 'A Sliced Wasserstein Loss for Neural Texture Synthesis," in *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021.

[16] Antoine Brochard, Sixin Zhang, and Stéphane Mallat. 'Generalized Rectifier Wavelet Covariance Models For Texture Synthesis," in *International Conference on Learning Representations (ICLR)*, April 2022.

[17] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. 'Generalized Rectifier Wavelet Covariance Models For Texture Synthesis," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, April 2019.

[18] Guilin Liu, Rohan Taori, Ting-Chun Wang, Zhiding Yu, Shiqiu Liu, Fitsum A. Reda, Karan Sapra, Andrew Tao, and Bryan Catanzaro 'Transposer: Universal Texture Synthesis Using Feature Maps as Transposed Convolution Filter," in *arXiv:2007.07243 [cs.CV]*, July 2020.

[19] Morteza Mardani, Guilin Liu, Aysegul Dundar, Shiqiu Liu, Andrew Tao, and Bryan Catanzaro. 'Neural FFTs for Universal Texture Image Synthesis," in *Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS)*, April 2020.