Texture synthesis: Feature Maps as Transposed Convolution Filter for decoding Self Similarity Maps

September 7, 2023

Antonio D'Orazio

Abstract

This project focuses on the problem of Texture Synthesis taking inspiration from the NVIDIA's Transposer paper. The objective is to reconstruct their method from scratch due to the unavailability of the code. To overcome hardware constraints, various alterations to the original approach have been made. For instance, I employ the Frechet Inception Distance (FID) instead of the GAN loss. Additionally, the project explores the usage of the Sliced Wasserstein Loss as a substitute for the style loss based on Gram matrices.

1. Introduction

My project addresses the problem of texture synthesis given a smaller and non-repetitive texture with a data-driven approach. The project is an implementation from scratch of the NVIDIA's Transposer paper with many modifications. An in-depth visualization of the metrics and the final results is available in the wandb.ai report. The code of my project is available in the github repository.

2. Related works

- (Liu et al., 2020): the original paper from NVIDIA from which I based my project on.
- (Heitz et al., 2021): they proposed a data-free approach to synthesize textures using a novel loss called the Sliced Wasserstein Loss, in substitution to the VGG-based Gram Loss.

3. Method

The network consists of an **encoder** layer, a **self-similarity map** (SSM) layer and a **decoder** layer. The encoder extracts features from the input image, which are passed to the SSM layer for computing the self-similarity between the extracted features. The map is computed at three different scales of the encoded result.
The decoder takes as input both the feature maps and the self-similarity maps. Then, instead of using trained weights for the transposed convolution layers, the weights and the biases are a transformation of the feature maps extracted from the encoder, while the input is the self-similarity map itself. In this way, the image features are slided across the self-similarity map. Such architecture simulates the traditional tiling operation which is useful to enlarge the pattern for synthesizing a texture. I implemented both the original Encoder, as proposed in the original paper, and a VGG-Based encoder.

4. Architecture

Encoder: The project implements two different encoders. The **Encoder** class is an implementation of the NVIDIA's paper. To improve the training speed and to minimize the hardware requirements, I created the **VGGEncoder** class which is a custom encoder based upon the pre-trained VGG as a feature extractor, and an additional Conv2d to upsample the layer to maintain compatibility with the rest of the network.

Self-Similarity Map: The self-similarity map is computed at three different scales, i.e., the feature maps received from the Encoder at three different levels. The self similarity map is defined by the following equation:

$$S(p,q) = -\frac{\sum_{m,n,c} \|F_c(m,n) - F_c(m-p,n-q)\|^2}{\sum_{m,n,c} \|F_c(m,n)\|^2} \tag{1}$$

Such equation can be easily approximated by convolutional layers with pre-defined weights.

Decoder: The decoder contains a Transposed Convolution which takes as input the Self-Similarity map, then it uses a transformation of the encoder output as filters. Finally, there is the traditional sequence of Convolution + Upsampling to finalize the output, with a BatchNorm2d after each convolution.

Email: Antonio D'Orazio <dorazio.1967788@studenti.uniroma1.it>.

Deep Learning and Applied AI 2023, Sapienza University of Rome, 2nd semester a.y. 2022/2023.

5. Loss functions

5.1. Style Loss

The Gram Matrix is the approach described from the original paper. The Sliced Wasserstein is the new approach taken from (Heitz et al., 2021).

Gram Matrix: The L1 distance of the Gram matrix of the VGG features extracted from the synthesized image and the (upscaled) original image. Computed with the features at levels {4, 9, 14, 23, 31}.

$$\mathcal{L}_{\text{style}} = \sum_{p=-1}^{P-1} \frac{\|K_p((\Phi_p^{I_{out}})^T(\Phi_p^{I_{tgt}}) - (\Phi_p^{I_{tgt}})^T(\Phi_p^{I_{out}}))\|_1}{C_p C_p} \quad (2)$$

Sliced Wasserstein: The Sliced Wasserstein loss is a textural loss that measures the distance between two feature distributions in a convolutional neural network. It is based on the Sliced Wasserstein Distance, which approximates the optimal transport distance by projecting the features onto random directions and sorting the projections. The loss is defined as the expected value of the L2 distances between the sorted lists for each layer and direction.

$$\mathcal{L}_{\text{SW}} = \mathbb{E}_V[\mathcal{L}_{\text{SW1D}}(p_V^l, p_{\tilde{V}}^l)] \quad (3)$$

Where L_{SW1D} is the Sliced Wasserstein Distance.

$$\mathcal{L}_{\text{SW1D}} = \frac{1}{|S|}\|sort(S) - sort(\tilde{S})\|^2 \quad (4)$$

This loss captures the complete feature distributions and is proven to converge to the target distribution. It can be used for texture synthesis, style transfer, and training generative neural networks.

5.2. Content Loss:

The L1 distance of the VGG features extracted from the synthesized image and the upscaled original mimage. Computed with the features at the last level before the classification layer.

5.3. FID Loss:

In replacement of the original GAN loss as stated in the paper. For hardware limitation, I'm using the Frechet Inception Distance (FID). The FID score quantifies the realism of generated samples, by measuring the Frechet Distance between the real and the fake image.

6. Training Details

The model was trained for **600 epochs**, with the **batch size** of **24** and **16**, respectively for the Gram and the Wasserstein losses, due to GPU limitations, and the **learning rate** set to **0.0032**. The learning rate decays by a factor of **0.1** every 150 epochs. The training was performed with a **GTX 1070 8GB**, with an average training time of **18 hours**. The dataset used is the **Describable Texture Dataset** (Cimpoi et al., 2014).

7. Experiments

The training was tested with four different variants of architecture and loss. **NVIDIA Encoder** represents the original encoder as it is described in (Liu et al., 2020). **VGG Encoder** is the variant I propose which is made by a VGG feature extractor and an upsample layer. **Gram** represents the style loss implemented with the distance between the Gram matrices of the VGG features. **Wasserstein** represent the style loss implemented by the Sliced Wasserstein Loss, as it's shown in (Heitz et al., 2021).

8. Results

Table 1. Performance comparison.

Models	SSIM	FID
NVIDIA Transposer (original)	**0.437**	**0.743**
VGG Encoder + Gram	0.159	1.565
NVIDIA Encoder + Gram	*0.164*	*1.558*
VGG Encoder + Wasserstein	0.161	6.578
NVIDIA Encoder + Wasserstein	0.159	8.702

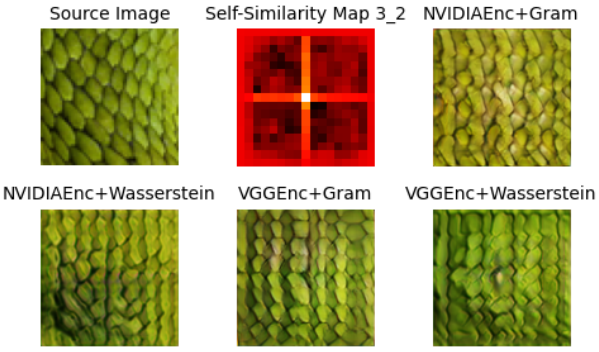

Figure 1. Comparison of the results.

9. Conclusion

Interesting results can still be reached with far less hardware resources, although sacrifying realism and pattern consistency, the reason being that the FID loss didn't prove to be a good replacement for the GAN loss. The pre-trained VGG network as the encoder had interesting results, with less memory usage but overall the NVIDIA Encoder still seems to perform better. The Sliced Wasserstein Loss was promising when coupled with the NVIDIA Encoder, while it shows more artifacts when used with the VGG encoder.

References

Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., , and Vedaldi, A. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

Heitz, E., Vanhoey, K., Chambon, T., and Belcour, L. A sliced wasserstein loss for neural texture synthesis. 2021.

Liu, G., Taori, R., Wang, T.-C., Yu, Z., Liu, S., Reda, F. A., Sapra, K., Tao, A., and Catanzaro, B. Transposer: Universal texture synthesis using feature maps as transposed convolution filter. 2020.