Texture Synthesis

January 1, 2024

Rigels Hysaj

Abstract

Texture synthesis is a fundamental process in computer graphics, enabling the generation of detailed surfaces from limited samples. Traditional methods often lead to repetitive and non-scalable textures, which are not suitable for dynamic or detailed environments. This project addresses the challenge of synthesizing high-quality textures from small samples using Generative Adversarial Networks (GANs), which promise seamless and scalable textures.

1. Introduction

Texture synthesis, a key aspect of digital graphics, involves creating textures for seamless application over large surfaces. In this project, two Generative Adversarial Networks (GANs) approaches were explored. Initially, a data-driven approach using the Describable Textures Dataset (DTD) (Cimpoi et al., 2014) was attempted, but computational resource constraints limited its success. To overcome this, I shifted to a data-free method, utilizing a single image for texture synthesis. This method, involving random cuts in the image to create new samples for GAN training, proved to be more resource-efficient and effective despite the limitations of the data-driven approach.

2. Related works

Texture synthesis has evolved significantly with the development of Generative Adversarial Networks (GANs). Traditional methods, often limited by artifacts like tiling or stretching, have been superseded by GANs, which can generate detailed textures without visible repetitions. Goodfellow et al.'s foundational work on GANs (Goodfellow et al., 2014) has opened new avenues in texture synthesis, further explored in Ulyanov et al.'s "Texture Networks" (Ulyanov et al., 2016), introducing efficient feed-forward networks for texture generation and style transfer. Perceptual loss, as introduced by Johnson et al. (Johnson et al., 2016), has also played a critical role in enhancing texture synthesis, focusing on structural similarity over pixel-level accuracy. Models like CycleGAN (Zhu et al., 2017) have enabled the translation of textural patterns across domains, and StyleGAN (Karras et al., 2019) has been instrumental in generating high-fidelity textures from small samples, scaling them up without repetition. Contemporary research continues to delve into both data-driven approaches, leveraging large datasets for learning, and data-free methods, using minimal samples for texture generation.

Email: Rigels Hysaj <hysaj.1706263@studenti.uniroma1.it>.

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

3. Methodology

The project began with a focus on a data-centric strategy, leveraging the Describable Textures Dataset (DTD) for the initial training phase. However, due to limitations in computational resources, it was necessary to adjust the methodology. Instead of utilizing the entire dataset, the training was conducted on selected portions, specifically targeting distinct texture patterns such as 'dotted', 'banded' and 'zigzagged'. Unfortunately, this targeted training approach did not yield the anticipated results. The model, when trained exclusively on 'zigzagged' patterns, failed to generate new images with the same texture. A similar outcome was observed with 'dotted' and 'banded' patterns. Consequently, this led to the adoption of a data-free learning approach. I began the new approach by training the model with a single image. To introduce variation and prevent overfitting, during each training iteration, a different random crop from this image was utilized. This method encouraged the model to learn and generalize the texture pattern more effectively, allowing for the generation of diverse texture patterns.

GAN Architecture The project utilized a Generative Adversarial Network (GAN) architecture to create textures. This GAN comprises a generator designed to produce textures and a discriminator to evaluate them. For training, random crops from original image were used to generate a diverse training set, which helped the model learn to produce varied textures.

Discriminator Architecture:

The Discriminator consists of a series of convolutional layers (‘nn.Conv2d‘). These layers progressively downsample the input image, extracting increasingly abstract features. The first layer takes an input with 3 channels (assuming RGB images) and expands it to 64 channels. Subsequent layers double the channels while reducing the spatial dimensions. After each convolutional layer, a Leaky Rectified Linear Unit (‘nn.LeakyReLU‘) is used as the activation function. The parameter 0.2 determines the slope for negative inputs, allowing a small gradient when the unit is not active, which can help mitigate the vanishing gradient problem. ‘nn.BatchNorm2d‘ is used after most convolutional layers to stabilize learning by normalizing the input to have zero mean and unit variance.

Generator Architecture:

The Generator begins with a convolutional layer (‘nn.Conv2d‘) with reflection padding, followed by ReLU activation. This layer expands the 3-channel input into 64 channels. It uses two convolutional layers to downsample the feature map while increasing the channel depth. A series of residual blocks are used. These blocks help in preserving the information through the network and alleviating the vanishing gradient problem. The Generator uses transposed convolutions (‘nn.ConvTranspose2d‘) to upsample the feature maps back to the original image size. The final layer is a convolutional layer that outputs the generated image with 3 channels (RGB).

Loss Function and Training The model employs a combination of Binary Cross-Entropy Loss (BCELoss) and Perceptual Loss to optimize both the discriminator and the generator. BCELoss is used to guide the discriminator in accurately distinguishing between real and generated images, while the generator is encouraged to create images that the discriminator will classify as real. Perceptual Loss, implemented via a pre-trained VGG-16 model, measures the perceptual distance between the features of generated and real images. This approach enables the model to capture the textures and fine details of images, emphasizing perceptual similarities rather than pixel-by-pixel differences. The loss function for the GAN can be summarized as follows: Discriminator Loss:

$$L_D = \frac{1}{2}\left(\text{BCE}(D(x), 1) + \text{BCE}(D(G(z)), 0)\right) \quad (1)$$

where D(x) is the Discriminator's output for a real image x, G(z) is the Generator's output for noise z, and BCE is the Binary Cross-Entropy Loss. Generator Loss:

$$L_G = \text{BCE}(D(G(z)), 1) + \text{PerceptualLoss}(G(z), x) \quad (2)$$

which combines the adversarial loss (trying to fool the Discriminator) and the perceptual loss (matching the generated image to the real image). This setup allows the GAN to not only generate images that look realistic (due to adversarial training) but also closely resemble specific target images in a perceptual sense (due to perceptual loss).

4. Results

The model achieved variable results depending on the complexity and nature of the textures. For textures such as tree's trunk, water, cracked, and zebra, the model generated satisfactory images that reflect the desired qualities without repetitions or stretching. Textures like grass, wood-floor, wall, and branches showed acceptable but improvable results, indicating a partial capture of the necessary features. The model was unsuccessful with the bubble, stars/galaxies and roof textures. Inside the 'Results' folder, there is a dedicated folder for each texture that I have chosen to experiment with. Inside each of these texture folders, you will find the original image, along with another folder that contains all the textures generated from that original image. Within the "Results" directory, there is also a subfolder named "data-driven" that holds the outputs from the models which were trained specifically on images featuring zigzagged and dotted patterns. However, as previously mentioned, the outcomes from this particular training exercise did not meet the expected standards of quality. The code and the results are linked below (git)

5. Discussion and Conclusions

In summary, limited resources constrained the use of the full Describable Textures Dataset (DTD), and even with DTD's subcategories, the model underperformed. However, the data-free approach yielded more promising results in most instances. Future efforts will concentrate on exploring diverse architectural designs to refine the data-driven method, aiming to achieve better consistency and quality. Additionally, enhancing the data-free approach to address specific textures where it previously underperformed will be a key area of focus. These advancements are expected to significantly improve the model's versatility and effectiveness in texture synthesis.

References

Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., and Vedaldi, A. Describing textures in the wild. 2014.

git. Texture synthesis. URL https://github.com/rigelshysaj/Texture_Synthesis.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 2014.

Johnson, J., Alahi, A., and Fei-Fei, L. Perceptual losses for real-time style transfer and super-resolution. *European Conference on Computer Vision*, 2016.

Karras, T., Laine, S., and Aila, T. A style-based generator architecture for generative adversarial networks. *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. Texture networks: Feed-forward synthesis of textures and stylized images. *International Conference on Machine Learning*, 2016.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. *IEEE International Conference on Computer Vision*, 2017.