# Texture Synthesis

September 9, 2023

**Hazem Dewidar**

## Abstract

Textures are widely used in computer graphics applications such as videogames to give more realism to walls, floors, skins and more. When the texture is small with respect to the surface it shall be applied on, it's either stretched or applied in many patches. This often gives birth to either **stretched textures along the surface or weird looking artifacts**. Creating large enough textures with artists is both time consuming and expensive. This project aims to mitigate this problem with the development of deep tools based on a **GAN** (Goodfellow et al., 2014).

## 1. Introduction

Deep learning tools are taking over the world, enabling the creation of tools that can decipher complex patterns and generate highly realistic content across a diverse array of fields, from face recognition (Deng et al., 2022) and natural language processing (Devlin et al., 2018) to medical diagnostics (Ronneberger et al., 2015) and artistic tasks (Zhu et al., 2020). It comes natural to want to use tools able to generate data in order to maximize efficiency when doing time consuming jobs, such as creating textures. If the texture is too small, it will be either stretched or applied in many patches creating an unpleasant looking effect (tiling). There are many methods used in videogames to avoid this effect when there aren't larger textures available such as adding random noise, using tiling offsets, using normal maps, etc... This project aims to synthesise textures starting from a small example to be fed to the generator. What we want from the generator is to capture the patterns on in an input given example and generate a larger texture. We can train the GAN using two main methods: let the generator learn on the same pattern (thus creating a specialized generator) or let it learn on a large scale data (thus creating a generic generator not exceeding at a particular type of texture pattern). Our GAN is composed of a generator and a discriminator. The generator used in this project is very similar to the one used in the cycleGAN paper (Zhu et al., 2020). We have an encoder that feed to several convolutional blocks an image of 256x256 pixel, the output is then passed to several residual blocks (He et al., 2015) and finally a decoder will generate an image of size 512x512 pixels. The image is then given as input to a patchGAN (Isola et al., 2018) which classifies small image patches as real or fake, enabling fine-grained discrimination and detail preservation. This work has experimented the same GAN architecture with different training methods and losses.

## 2. Related Works

When GANs were first introduced, they got a lot of focus over the years with new architectures and applications being made. In the same year, Conditional GANS (Mirza & Osindero, 2014) were introduced in which we condition the network to some kind of data instead of random noise. The pix2pix (Isola et al., 2018) model was a type of conditional GAN that learns a mapping from am image domain $X$ to an image domain $Y$, effectively translating the images. One of the biggest contribution made by this paper was its discriminator model: patchGAN. They provide fine-grained discrimination, which is particularly useful for tasks that require detailed output. Additionally, because they focus on smaller patches, they help prevent mode collapse (where the generator produces only a few distinct outputs) that can occur in traditional GANs. This propriety is particularly useful when dealing this textures generation This architecture had one big limitation: it required paired data. Later on, CycleGAN (Zhu et al., 2020) were introduced that overcome this limitation. They do not need paired data but they only need examples from both image domains and two discriminators and generators. For texture generation, my work explored different training modalities and one of them (see 4.3) turned out that the idea was very similar to what (Zhou et al., 2018) proposed. A SOTA model is texture-diffusion. Given a small text prompt, it's able to generate a new texture.

## 3. Texture Synthesis

### 3.1. GAN: Generative Adversarial Network

#### 3.1.1. Generator

In this architecture, the generator is conditioned on a small texture of size 256x256 pixel and it will generate an image

of size 512x512 pixel. The generated image, ideally, is the expansion of the given image. This work uses, in addition to classic GAN loss $\mathcal{L}_{GAN}$, the $\mathcal{L}_2$ loss function between the generated image and the target expansion image (more detail in subsection 3.2). Some models had a third loss, the $\mathcal{L}_{Content}$. This is the L2 distance between the feature computed by a frozen pretrained ResNet-50 of the generated image and the target expansion. Ultimately, the final loss is:

$$\mathcal{L}_{Generator} = \mathcal{L}_{GAN} + \beta\mathcal{L}_2 + \gamma\mathcal{L}_{Content} \quad (1)$$

Basically the $\mathcal{L}_{GAN}$ encourages the generated output to be indistinguishable from real data according to a discriminator network. The $\mathcal{L}_1$ promotes the generator to produce outputs that are close to the ground truth in terms of pixel values. Finally the $\mathcal{L}_{Content}$ encourages the generator to capture high-level content and structural information and style from the target domain. The input image *X* is passed through an initial convolutional layer (Conv + ReLU) to compute filters while maintaining the initial image size. Then there are several convolutional blocks (Conv + Instance InstanceNorm2d + Relu) that downsample the tensor size while computing new filters. The output is then passed to residual blocks and then there is a skip connection that concatenates the output with the first input of the residual blocks. Finally, there are several Convolutional Blocks that upsamples the tensor and a final layer that generates a three channel image of size 512x512

#### 3.1.2. Discriminator

The type of the discriminator is PatchGAN. It works on local image patches rather than the entire image. This approach allows the discriminator to provide feedback at a more fine-grained level, providing better guidance to the generator during training. In the PatchGAN discriminator, the output of the network is a grid of probability values that indicate the discriminator's confidence in the authenticity of each local image patch. The generator is then trained to produce realistic textures that can fool the discriminator at the patch level (thus it is encouraged to generate a good texture everywhere). During training, the dataset module will return, besides the input to the generator, also the target expansion image and another random image. Randomly one was chosen among the two and the generator had to classify it in order to compute its loss.

### 3.2. Training Data

This model uses self-supervision for learning. Self-supervision is a training technique in which there is no need for labelled data. This approach is particularly useful in scenarios where obtaining labeled data is expensive, time-consuming, or simply not feasible. In particular, the image is cropped randomly by 512x512 pixel creating what I referred to as the **target expansion image**. Finally, it's center cropped by 256x256 pixel and added gaussian noise with probability 50%. The goal is to condition the generator on the cropped image to generate the expanded image. A small subset of the training data of (Cimpoi et al., 2014) was used for the Generalized Generator and Specialized Generator.

## 4. Approach and Experimental Results

The generated textures are linked below (git).

### 4.1. Generalized Generator

The generator is trained on a large quantity of images (over 5.500 images) with many patters. The goal is to see whether the generator is able to generalize on a large quantity of patterns. Due to low computational resources available (requires 10min per epoch), the model had not converged and it produces images dominated by the most common color of the input image. I expect that, upon convergences, the image will act as indexer and the model will generate a variation of the input.

### 4.2. Specialized Generator

The generator is trained on a small set of images conceptually representing the same pattern. Given an image with similar pattern to the one that it is trained on, it can generate unseen images. It can even do some basic linear interpolation between images in the training set. The input image will act as an indexer. This model has shown that, if given random noise, it will generate an unsatisfactory image.

### 4.3. Highly Specilized Generator

This is an extreme version of the Specilized Generator. One high resolution image is used during training which will be randomly sampled. This model is able to generate the same texture patterns even if it's given only random noise. If we give a random crop of the training image plus random noise, it is able to generate a variation of it.

## 5. Conclusions and future works

In this work, different training methods were explored. The introduction of $\mathcal{L}_{Content}$ led the model to generate better textures. For generating textures, the 4.3 training approach is more satisfactory. Given sufficient computational resources, future work can be done improve further the results. The resulting images do not contain the pattern's small details. To mitigate this problem, another network will be trained that given a synthetic texture, it is able to add details, along the lines of what SRGAN did (Ledig et al., 2017). Also the generalized Generator could be trained for many more epoches.

## References


Project repository. URL https://github.com/THESHADOW2030/DeepTextureGeneration.

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.

Deng, J., Guo, J., Yang, J., Xue, N., Kotsia, I., and Zafeiriou, S. ArcFace: Additive angular margin loss for deep face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44 (10):5962–5979, oct 2022. doi: 10.1109/tpami.2021.3087709. URL https://doi.org/10.1109%2Ftpami.2021.3087709.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL http://arxiv.org/abs/1810.04805. cite arxiv:1810.04805Comment: 13 pages.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks, 2014.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-image translation with conditional adversarial networks, 2018.

Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., and Shi, W. Photo-realistic single image super-resolution using a generative adversarial network, 2017.

Mirza, M. and Osindero, S. Conditional generative adversarial nets, 2014.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation, 2015.

Zhou, Y., Zhu, Z., Bai, X., Lischinski, D., Cohen-Or, D., and Huang, H. Non-stationary texture synthesis by adversarial expansion. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 37(4):49:1–49:13, 2018.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.