Adversarial Texture Synthesis with Prototypes

Nkosinathi HLophe*
African Masters in Machine Intelligence
African Institute for Mathematical Sciences Kigali, Rwanda
nhlophe@aimsammi.org

Lucas Theis
GoogleBrain, Berlin
theis@google.com

Abstract

The use of optimization based and generative adversarial network (GANs) based methods for texture synthesis have produced results which a human can hardly distinguish from real and generated textures. However the existing methods are either slow or unstable during training, or they require a lot of hyper-parameter tuning to produce near-perfect results. In this work, we focus on combining the advantages of the optimization-based and generative methods to come up with a faster and more stable prototype-based method. In the experiments, we show that our model can synthesize large textures from a smaller reference texture patches. We also discuss some limitations of our method with respect multi-texture synthesis.

1 Introduction

Textures are described as homogeneous images consisting of repeated elements, often subject to some randomization in their location, colour, orientation, size, etc [24]. Texture synthesis is the expansion of small reference texture examples to arbitrary large images while preserving the structural content of the original reference texture. To generate new texture images the spatial summary statistics of the generated texture should be matched with those of the reference texture.

For texture synthesis that is faster to optimize and more stable during training we propose a prototype-based texture synthesis which is a combination of optimization [9] and generative [10] approaches. Our method combines fast convergence advantages of GANs and the stable training process and variability of optimizations based methods. Generation of 3D textures and large textures can be a challenge for VGG-based feature extractor which relies on a specific input dimension. The prototype-based method allows for texture generation of higher dimensions and textures of any size without changing the architecture of the generator.

2 Related Work

Julesz [14] hypothesized that the N-th order joint histograms of image pixel can statistically characterize textures, many different statistical measures have been proposed (see [9] VGG based). Two images with similar spatial statistics are perceived to be from the same texture image [34], this claim was disputed by Julesz et al.[15] by producing a pair of different textures with identical second and third-order statistics. The quality of generated textures images is said to be successful if a human observer can not tell the difference between reference texture and generated ones [34].

There are two main categories of textures synthesis: generative and iterative approaches [2, 18]. Iterative texture synthesis examples include assemble-base, pixel-based [6], appearance-based [17], optimization-based [24] methods. These iterative methods can be slow and can struggle to generate

*This work was done as part of an MSc project

Preprint. Under review.

complex textures since they rely on matching global statistics between the reference texture and the output by optimizing some defined objective function [22]. The optimization-based method proposed by Kwatra et al.[16] uses a Markov Random Field (MRF) similarity metric for measuring the quality of generated textures, this allows the formulation of a minimization problem which uses expectation-maximization. All these methods they learn to generate textures using an iterative process, with the assumption that the loss function used to compare the synthesized textures and the reference texture is valid throughout the training process.

There have been recent developments in the use of deep convolutional neural networks (CNN) to learn texture features. Gatys et al.[9] utilizes learned features from a VGG-19 network that matches the Gram matrices between synthesized and reference textures using a predefined loss function. The work proposed by [9] is considered expensive to optimize which is why alternative methods have been proposed. One of the methods is the use of a deep feed-forward network to produced multiple diverse textures of any size using a single network [33]. This network use noise and a selection unit and input into the generator [18]. Another CNN-based method uses a spatial tensor as input to GAN that produces high quality and very scalability textures with respect to the output texture size [13].

Generative Adversarial networks have shown that they can generate images that look real [10]. In texture synthesis, adversarial approaches have been used by Jetchev et al.[13] that uses a fully convolutional architecture. The fully convolutional architecture allows the synthesis of textures with multiples sizes. The bimodal modal Field-of-Expert (BiFoE) which is an extension of Field of Expect (FoE) model is a generative model that generate better visual textures when trained with a better approximation of the likelihood gradient [11]. The use of recurrent image model which is based on multi-dimensional long short-term memory (LSTM) have shown to outperform state of the art model in quantitative comparison on several datasets and it has also shown some promising results in texture synthesis and image inpainting [32]. User controllable textures synthesis models that are based on GANs, which allows the user to choose the type of texture they want to produce [2]. Instead of using a fixed classifier or feature extractor, the GAN based texture synthesizer learn to extract better feature while at the same time it learns to generate textures that look real. This approach is harder to optimize since two networks (the generator and discriminator) have to be optimized simultaneously. To synthesize textures, the generator is initialized with random noise images and gradually learn to produce textures with similar statistics to that of the real textures images through adversarial learning.

In this work, we explore a combination of the optimization-based and the GAN-based methods. Instead of using a CNN based generator, we use a prototype which is faster, does not have too many hyper-parameters to tune and it is easier to choose the prototype structure architecture compare to choosing a generator architecture. GAN based methods have a problem with stability during training and can suffer from mode collapse [23], which can reduce the variability of the generated textures. Our texture synthesis method tries to take advantages of both the slow iterative based [8] and the faster GAN based method. With the right optimization, Ustyuzhaninov et al.[34] showed that a simple single-layer CNNs with random filters can be used as a basis for texture synthesis.

3 Method

Our model provides a novel way of texture synthesis that utilized both the optimization-based [9] and the GAN-based approaches of texture synthesis. To generate the texture we start with a 3D tensor prototype of size $3 \times W \times H$ where W, H are the spacial dimensions of the texture we want to generate. From the full prototypes, we randomly crop a smaller patch $x \in \mathcal{R}^{3\times w\times h}$ which is the size of the reference texture we want to learn. The cropped prototype texture is passed through a convolutional neural network (CNN) as shown in Figure 3 before going through the discriminator. The reference texture is fed to the discriminator to classify it as real.

To generate new textures from the reference texture, we iteratively optimize the discriminator's ability to label fake samples as fake and the real samples as real while the prototype tries to fool the discriminator into labelling the generated images from the prototype as real. The choices of the loss function play an important role when it comes to the quality of the textures that we can produce. Using Lipschitz regularization on the popular GANs (NSGAN [10], LSGAN [21], WGAN [3]) performs when the discriminator is regularized with a small Lipschitz constant [25]. The learning process of GANs is to train a discriminator and a generator simultaneously. The generator (in our case the prototype) tries to learn the distribution p_g over the data $\mathbf{x}$. The learned prototype tries to

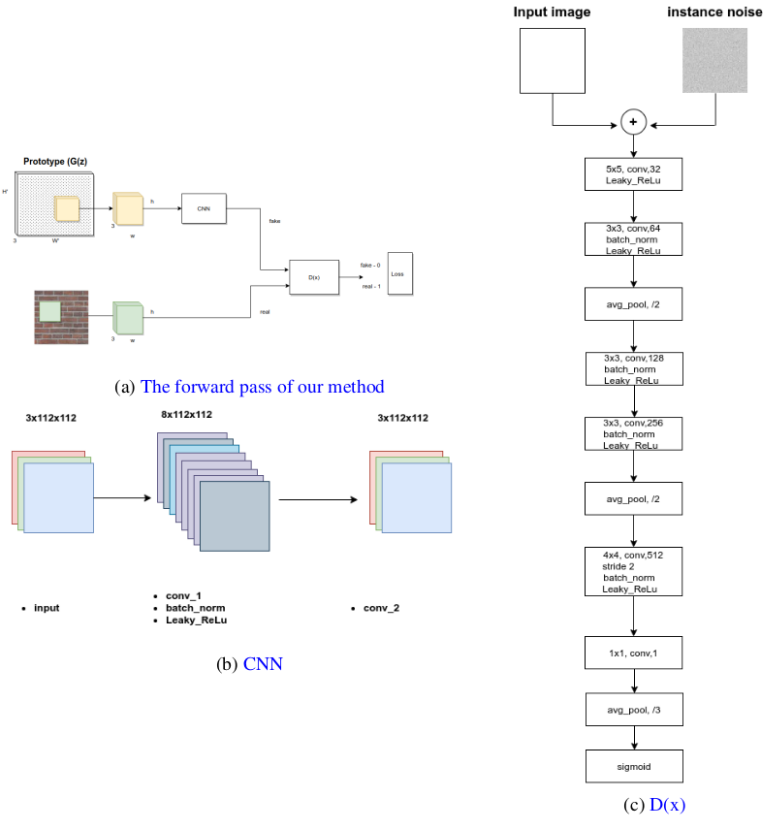

Figure 1: Training framework of the proposed method. (a) Diagram of texture synthesis pipeline from the initialization of the prototypes weights showing the forward pass. (b) The CNN architecture that is used in the prototype framework. (c) A fully convolutional discriminator architecture that learns to separate real textures from generated ones.

estimate

$$p_g(x) = \frac{1}{N}\sum_{n=1}^{N}\delta(x - \hat{x}_n) \tag{1}$$

, where $\hat{x}_n$ are the generated samples. Equation 2 below shows a hybrid approach which uses a degenerated prior in the generator $g_\theta(z)$ instead of a noise prior that is used in the standard GAN.

$$p_g(x) = \frac{1}{N}\sum_{n=1}^{N}\delta(x - g_\theta(\hat{z}_n)) \tag{2}$$

$$\tag{3}$$

The discriminator acts as a classifier that detects whether samples are fake or real. The objective function of the GAN is a minimax and can be expressed mathematically as

$$\min_G \max_D V_{GAN}(D,G) = \mathbb{E}_{\mathbf{x}\sim pdata(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim pg(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{4}$$

This GAN objective function has a problem with the gradient signal at the start of the training phase. To improve the gradient signal, the authors also propose the non-saturating, where the generator aims to maximize the probability of generated samples being real [20].

The improved non-saturation GAN (NSGAN) is expressed mathematically as

$$\min_D V_{NSGAN}(D) = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim pdata(\mathbf{x})}[\log\,(D(\mathbf{x}))] - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim pz(\mathbf{z})}[\log\,(D(1-G(\mathbf{z})))] \quad (5)$$
$$\min_G V_{NSGAN}(G) = -\frac{1}{2}\mathbb{E}_{\mathbf{z}\sim pz(\mathbf{z})}[\log\,(D(G(\mathbf{z})))].$$

The JS-divergence is a constant for our method when using the standard GAN loss. This is a problem because as the discriminator get better, that is $D(x) \approx 1 \forall x \in p_{data}$ and $D(x) \approx 0 \forall x \in p_g$ the loss function is almost zero meaning we end up having vanishing gradient [36]. The LSGAN proposed by Mao et al.[21] minimizes the Pearson $\mathcal{X}^2$ divergence. The LSGAN loss in Equation 6 takes the same form as the non-saturation GAN from [10]. The benefits of the LSGAN over the ordinary GAN loss is that it penalizes correctly classified samples that are far away from the decision boundary which can prevent the problem of vanishing gradient. This allows the LSGAN to be more stable during the learning process [21].

$$\min_D V_{LSGAN}(D) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim pdata(\mathbf{x})}[(D(\mathbf{x})-1)^2] + \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim pz(\mathbf{z})}[(D(G(\mathbf{z})))^2] \quad (6)$$
$$\min_G V_{LSGAN}(G)' = \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim pz(\mathbf{z})}[(D(G(\mathbf{z}))-1)^2].$$

In adversarial training, both the generator and the discriminator are updated iteratively to optimize the objective function. The DCGAN provided a more stable architecture for training GANs to produce images both in supervised and unsupervised learning [27]. This network also gives good representations of images for generative modelling. The discriminator architecture that we used followed some of the guidelines suggested by [27]; use batchnorm, remove fully connected hidden layers, use LeakyReLU as activation. Also, instead of max-pool using we use average-pool as [34] suggested that it produce better results. The discriminator architecture used in all our experiments is shown in Figure 3(c). The discriminator can be made to have a single output or multiple outputs by simple using global average pooling across each feature map [19]. This technique is more stable and memory efficient as compare to using fully connected layers.

4 Experiments

The performance of our non-parametric prototype-based texture synthesis with data from natural textures and compared with state-of-the-art texture synthesis models. Following Ustyuzhaninov et al.[34] and Mardani et al.[22], we use textures from [4, 1, 7, 26, 5] datasets.

4.1 Network architecture and training

The architecture for the 256×256 texture from 128×128 input is as follows. Start with a $3\times384\times384$ prototype with weights initialized to zero mean and 10^{-10} std. Next, randomly crop a $3 \times 112 \times 112$ from the prototype then feed it to the 2 convolutional layer network in Figure 3(b). The outputs of the first and second conv layers are $8 \times 112 \times 112$ and $3 \times 112 \times 112$ respectively, both with same padding and filter size of 3×3. To get $3 \times 256 \times 256$ output we simple crop it from the $3 \times 384 \times 384$ after training. The weights of the convnets are initialized to zero and the bias of the last layer are initialized to the mean channel of the reference texture we want to learn.

Discriminator takes in 16 patches of $3 \times 112 \times 112$ randomly selected from the prototypes and 16 randomly cropped patches from the $3 \times 128 \times 128$ texture. The inputs are translated into outputs through a series of 5 conv layers and average pooling with batchnorm and LeakyReLu applied in every layer Figure 3 (c). The last two layers are a 1×1 conv2d and global average pooling which is used to determine the output size of the discriminator network. For all the texture generation, the model was trained for 30 epoch with 200 iterations per epoch. During training, we add instance noise ($\mathcal{N}(0, 1)$) into both the input images (x_{in}) and generated images before we feed them into the discriminator.

$$x_{in} = x + \sigma\mathcal{N}(0, 1) \quad (7)$$

The instance noise helps stabilize the learning of both the generator and discriminator by minimizing the divergence between p_{data} and $pz(\mathbf{z})$ [31, 13]. The added input noise is multiplied by a noise factor ($\sigma \in [0, 1]$) before adding them into the input image shown by equation 7. The noise factor starts at 0.5 and reduces by a factor of 4 every 10 epochs.

4.2 Quality metrics

Before presenting the experiments, we will introduce the quality measurement of the images produced by the prototype. We use two quantitative metrics; A VGG model based loss and FID based on inception v3 model.

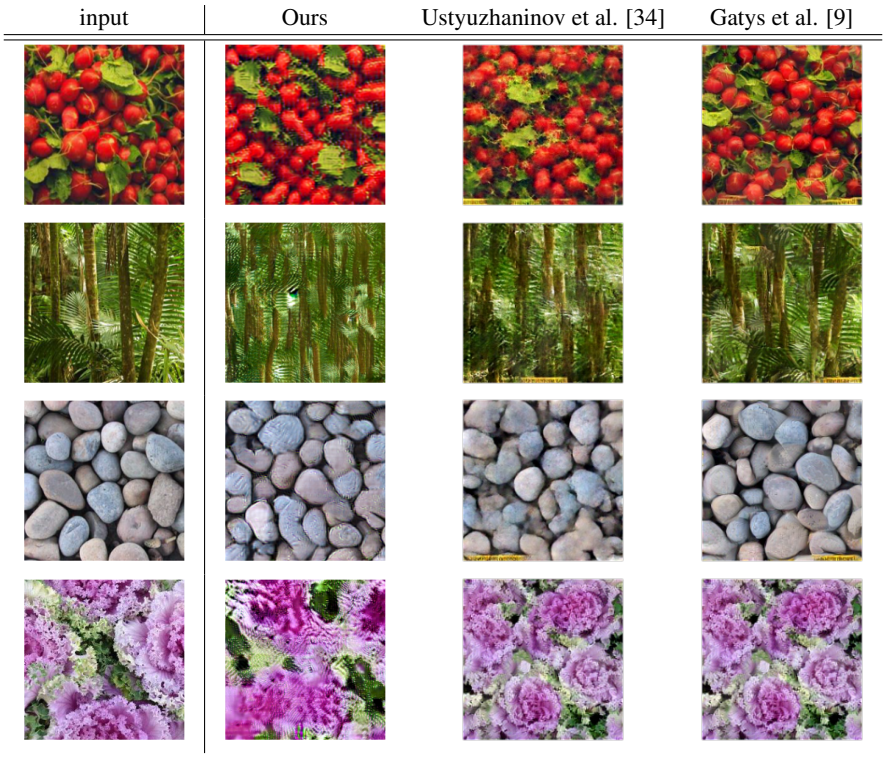

Figure 2: The reference texture is shown in the first column and three samples that were synthesized from our method, [34] and [9] all of them with size 256×256.

First the VGG loss based on the ReLU activation layers of the pre-trained 19 layer VGG network described in [29]. The activations from the feature maps of the VGG are stored as a matrix $F^l \in \mathcal{R}^{N_l \times M_l}$ where N_l is the number of filters in layer l and M_l is the vectorized feature maps per filter per layer. From the feature maps, we compute the Gram matrix $G^l \in \mathcal{R}^{N_l \times N_l}$ which is inner produce between the feature maps

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}. \tag{8}$$

We use the gram matrix from each layer activation to compute the contribution of each layer to the final loss

$$E_l = \frac{1}{(\sum_{i,j} G^l_{ij})^2} \sum_{i,j} (G^l_{ij} - \hat{G}^l_{ij})^2, \tag{9}$$

and the total loss is the sum of the losses from each of the 5 layers from the activations

$$\mathcal{L}(x, \hat{x}) = \sum_{l=0}^{5} E_l \tag{10}$$

The second quantitative method is the Frechet Inception Distance (FID) which was proposed by [12] to evaluate the quality of generated images in GANs. Just like the VGG based loss, this one is based on a pretrained model this time the Inception v3 (ϕ) [30]. The model ϕ is used to get a 2048 feature vector which is the prediction from the images. For each pair of generated and real texture images using the feature vectors, we calculate the means (μ) and covariance matrices (Σ). The FID matrix is given by:

$$FID = ||\mu_r - \mu_g|| + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}), \tag{11}$$

where Tr is the trace. Since we want visually appealing generated images, human evaluation is important.

4.3 Results

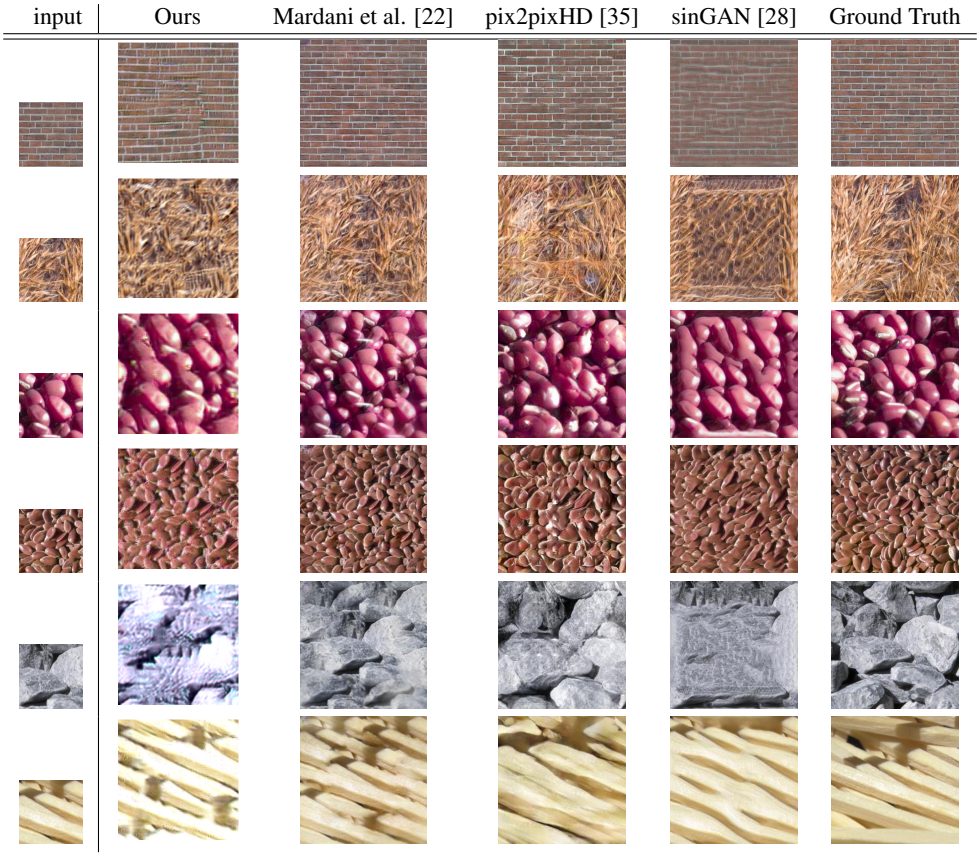

Figure 3: Each row shows the reference texture (left) and 6 samples that were synthesized from different methods for $(128 \times 128) \rightarrow (256 \times 256)$.

First, we show texture generated with our prototype method in Figure 4.2 and compare with [34] and [9]. Here we show four different images. The first column from Figure 4.2 shows different reference textures. The generated images used for comparing were obtained from [34]. We observed that our prototype method was able to learn to generate other textures but struggled to fully reproduce the texture of trees from row 2 of Figure 4.2. The results produced by our methods is impressive considering that all inductive biases are in the loss and the generator provides no inductive biases.

Figure 4.3 shows the results of texture synthesis from 128×128 input to 256×256 output. The results show textures generated by our model with models from [22], [35], and [28]. The images used for comparing in Figure 4.3 were obtained from [22]. These results show that our methods can produce results that are visual comparable with the texture generated by state-of-the-art methods.

The quality metrics results are presented in Table 1. To come up with the results in the table we took 50 random crops from the reference textures and the generated textures, compute the losses using the FID and vgg loss methods and took the average. This gave us the average loss per texture for each model, to get the results presented we repeated the above steps for 8 different textures. For the FID our models show results that are comparable to [28] and [34]. For the vgg loss, our method outperforms [28] and [34].

Table 1: Performance of different texture synthesis methods from 6 sample that were sampled multiple times

	FID	Vgg loss
Gatys et al. [9]	138.8	1.20
Ustyuzhaninov et al. [34]	252.4	4.79
Mardani et al. [22]	**103.0**	**0.947**
pix2pixHD [35]	193.7	1.62
sinGAN [28]	227.4	4.74
Ours	286.6	3.94

5 Discussion

We introduced an iterative texture synthesis method based on adversarial training. Our method learns to generate textures of any size from a small patch of the reference texture. Initialization of the prototype plays an important role in producing visually impressive results. Random initialization of the weights of the prototypes lead to late convergence during training or at worse the method does not produce meaningful textures. Also, the introduction of the CNN network in the prototype improves the stability of the model during training. The introduction of the noise helped reduce the artifacts in the texture generated with our method and make the model converge to better visually generated textures. We can generate larger texture than the $128 \rightarrow 256 \times 256$ all we need to do is increase the size of the prototypes before training.

The main drawback of our method is in the non-symmetric edges of the prototypes which require the training to be done on a bigger size than the required shape then crop the size we want. This can be compensated by how fast and easy it is to increase the size of the prototypes before training. To produce different textures we need to train multiples models.

This paper we focus on combining the advantages of optimizations based and generative based methods using adversarial training on prototypes to learn the reference texture. Given a small texture sample, our proposed method can synthesize textures of any size using a small reference texture.

References

[1] *Center for Machine Vision Research. Outex texture database.*, 2020 (accessed November 27, 2020). `https://computervisiononline.com/dataset/1105138685`.

[2] A. Alanov, M. Kochurov, D. Volkhonskiy, D. Yashkov, E. Burnaev, and D. Vetrov. User-controllable multi-texture synthesis with generative adversarial networks, 2019.

[3] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan, 2017.

[4] G. J. Burghouts and J.-M. Geusebroek. Material-specific adaptation of color invariant features. *Pattern Recognition Letters*, 30(3):306–313, 2009.

[5] D. Dai, H. Riemenschneider, and L. Van Gool. The synthesizability of texture examples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3027–3034, 2014.

[6] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1033–1038. IEEE, 1999.

[7] S. Fekri Ershad. A new benchmark dataset for texture image analysis and surface defect detection, 06 2019.

[8] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style, 2015.

[9] L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks, 2015.

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

[11] N. Heess, C. Williams, and G. Hinton. Learning generative texture models with extended fields-of-experts. 01 2009. doi: 10.5244/C.23.115.

[12] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[13] N. Jetchev, U. Bergmann, and R. Vollgraf. Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv:1611.08207*, 2016.

[14] B. Julesz. Visual pattern discrimination. *IRE Trans. Inf. Theory*, 8:84–92, 1962.

[15] V. J. Julesz B, Gilbert EN. Visual discrimination of textures with identical third-order statistics. *Biol Cybern*, pages 137–40, 1978. doi: 10.1007/BF00336998.PMID:728493.

[16] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. Texture optimization for example-based synthesis. *ACM Trans. Graph.*, 24:795–802, 07 2005. doi: 10.1145/1186822.1073263.

[17] S. Lefebvre and H. Hoppe. Appearance-space texture synthesis. *ACM Transactions on Graphics (TOG)*, 25(3):541–548, 2006.

[18] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang. Diversified texture synthesis with feed-forward networks, 2017.

[19] M. Lin, Q. Chen, and S. Yan. Network in network, 2014.

[20] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet. Are gans created equal? a large-scale study, 2018.

[21] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks, 2017.

[22] M. Mardani, G. Liu, A. Dundar, S. Liu, A. Tao, and B. Catanzaro. Neural ffts for universal texture image synthesis. *NeurIPS 2020*, 2020.

[23] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks, 2017.

[24] J. Portilla and E. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40, 10 2000. doi: 10.1023/A:1026553619983.

[25] Y. Qin, N. Mitra, and P. Wonka. How does lipschitz regularization influence gan training?, 2020.

[26] P. M. R. Kwitt. Salzburg texture image database.

[27] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[28] T. R. Shaham, T. Dekel, and T. Michaeli. Singan: Learning a generative model from a single natural image, 2019.

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision, 2015.

[31] C. K. Sønderby, J. Caballero, L. Theis, W. Shi, and F. Huszár. Amortised map inference for image super-resolution, 2017.

[32] L. Theis and M. Bethge. Generative image modeling using spatial lstms, 2015.

[33] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images, 2016.

[34] I. Ustyuzhaninov, W. Brendel, L. A. Gatys, and M. Bethge. What does it take to generate natural textures? In *ICLR*, 2017.

[35] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans, 2018.

[36] L. Weng. From gan to wgan, 2019.