Multiclass texture synthesis using generative adversarial networks

Bc. Maroš Kollár[*]
Supervised by: Dr. Lukáš Hudec[†]

Faculty of Informatics and Information Technologies
Slovak University of Technology
Bratislava / Slovak republic

Abstract

Generative adversarial networks as a tool for generating content is currently one of the most popular methods for content synthesis. Despite its popularity, current solutions suffer from a drawback of shortage of generality. It means that trained models are usually able to synthesize only one specific kind of output. The usual texture synthesis approach for generating N different texture species requires training of N models with changing training data. In our work, we present a synthesis architecture model constraining and forcing the optimization for generating multiple texture types. We focus on the synthesis of realistic natural non-stationary textures. The solution allows users to control the class of texture to synthesize. Thanks to the controllable selections from feature space of synthesized texture, we also explore the possibilities of transitions between classes of trained textures for potential better usage in applications where texture synthesis is required.

Keywords: texture, GAN, multiclass, synthesis

1 Introduction

The definition of texture highly depends on the application area, in which we use this term [10]. Nevertheless, in general, textures describe surface properties of objects like appearance, structure, consistency, or feeling from touch. Textures are an essential component of computer vision because of their usage in tasks like classification, detection, or segmentation used in medicine or the technical industry. Textures are also essential for the graphics and entertainment industry since almost every animated movie, video game, or other product depends on its visual appearance.

We can sort textures based on their primary characteristic into groups like smooth, rough, glossy, matte, et cetera. There are also more general characteristics like stationarity and homogeneity [25, 18] that profile textures. It is possible to say that both of these features describe an aspect of texture complexity. Stationarity represents the regularity of structure. Homogeneity represents how many elementary textures are included in the evaluated texture. The more complex the texture structure is, the more difficult it is to synthesize it.

Texture synthesis is a process of creating artificial textures that can be used to augment datasets needed for computer vision tasks or to replace texture photographing or painting with a more comfortable and less time-consuming method in the process of content creation. There are multiple texture synthesis approaches; however, current research orients on Generative adversarial networks (GANs). GANs proved their advantages in quality of outputs and speed of generating. However, their disadvantages are long and challenging training accompanied by problems like vanishing gradients or mode collapse. Another drawback is that current solutions are trained to synthesize one texture class, and multiple learned models are required to synthesize multiple texture classes. That results in higher disk storage requirements for storing the network models and inability of transitions between individual textures.

In this paper, after a general overview of image synthesis approaches, we describe the architecture and training of our method for synthesizing multiple texture classes by one model focused on synthesizing non-stationary textures.

We built our approach focused on multiclass generation based on a generator proposed by work of Li et al. [14], and network inputs inspired by Bergmann et al. [4] and Li et al. [14] with our modification of obtaining texture type information by pretrained classification network. To ensure stable training and prevention from mode collapse, we used approaches of progressively growing GAN [11] and minibatch discrimination [21]. We also show the results of the proposed method, explore the possibility of transition between texture classes and propose a promising approach with the siamese neural network as texture type extractor for further improvement of our method.

[*]maroskollar2@gmail.com
[†]lukas.hudec@stuba.sk

2 Related work

Texture synthesis has been an active field of research for multiple decades. Many different approaches were introduced and categorized to groups during these years based on their main feature.

Non-parametric sampling is considered a traditional approach and, for a long time, was one of the most popular synthesis methods. This approach uses copying parts of sample textures to create a new one. Parts of sample textures are chosen based on the similarity of their neighborhood and area of an already synthesized part of the texture. There are two main types of non-parametric sampling based on the size of individual parts copied to synthesized texture. Pixel-based synthesis [6, 24, 22, 3] that creates texture pixel by pixel and patch-based synthesis [19, 15, 13] that copies whole patches. These approaches are intuitive and relatively easy to implement. On the other hand, their synthesis is quite slow, and there is a possibility that synthesized textures will contain visually duplicate parts.

In contrast to non-parametric sampling, parametric synthesis [18] uses parameters to describe texture statistics. The synthetized image is created by gradually changing random noise [7]. Two textures should have identical statistics to be considered similar [16].

In recent years textures have been synthesized mainly by using neural networks. Gatys et al. [7] created a parametric approach that used convolutional neural network VGG-19 [23] to extract Gram matrices at multiple layers as texture statistics. The new texture is synthesized from random noise. Noise is passed through the network and edited by gradient descent to minimalize the difference between Gram matrices of example texture and new texture.

Variational autoencoders [12, 5, 17] are another approach that uses neural networks to synthesize textures. Variational autoencoders are similar to autoencoders, but their function is to create similar output, not identical. They consist of an encoder part that maps input data to low dimensional representation and a decoder part that reconstructs this representation to output. A drawback of this synthesis solution is the quality of output that could be blurry.

Currently, the most popular approaches of texture synthesis are based on generative adversarial networks introduced by Goodfellow et al. [8]. Generative adversarial networks contain two neural networks, discriminator and generator, that train themselves by min-max two-player game. That results in improved output quality of generated output. The generator's goal is to use random noise to create output that the discriminator would not reveal as fake. The goal of the discriminator is to determine which inputs are real and which are fake correctly. Since the original GAN solution introduction by Goodfellow et al., multiple GAN modifications have been created. Proposed modifications changed architecture of generator or discriminator [20], used different adversarial loss functions [2, 9], stabilized training [11] or create new approach of training [26]. A few GAN solutions also focused on multiclass texture synthesis [14, 4, 1] to synthesize multiple texture classes with one model.

3 Method

The main goal of our work is to created a robust method for multiclass texture synthesis with a focus on non-stationary textures and maintaining the quality of generated textures. The architecture of our solutions is shown in figure 1.

3.1 Generator

The core of our approach, generator, is based on the work of Li et al. [14]. The architecture of the generator network is constructed with two streams that help to synthesize the required type of texture. The primary stream handles the synthesis of the final texture. The secondary stream processes information of which class of texture should be generated. Activations of secondary streams are merged as 32 channel image to activations of the primary stream after every processed spatial upsample. This ensures that the primary stream has additional information about a class of synthesized textures at every resolution. Upsampling of resolution in both primary and secondary streams is done by upsample function with scale factor 2. The exception is the first upsample of vector done by transposed convolution. After upsample, activations are processed by a set of convolutional layers with instance normalization and leaky relu activation functions.

Because of the difficulty of multi-class non-stationary texture synthesis, the solution suffered from visual artifacts and had a problem learning all types of presented textures even though there were only six. To deal with this problem, we implemented the generator as progressively growing GAN [11]. This approach helped to stabilize learning thanks to the ability of the network to firstly learn the color palette of synthesized textures and then increase resolution and learn details of textures. The current implementation of our solution generates outputs of 128×128 pixels. However, thanks to progressive growing GAN implementation, more layers can be easily added to increase the resolution of the final output. We also changed batch normalization to instance normalization and added hyperbolic tangent as the final layer to improve the quality of outputs. Hyperbolic tangent limits values of output pixels between -1 and 1 to prevent very high or low values. The generated output is then used to train the discriminator without clipping to [0, 1] to force the generator to learn the correct interval of pixel values.

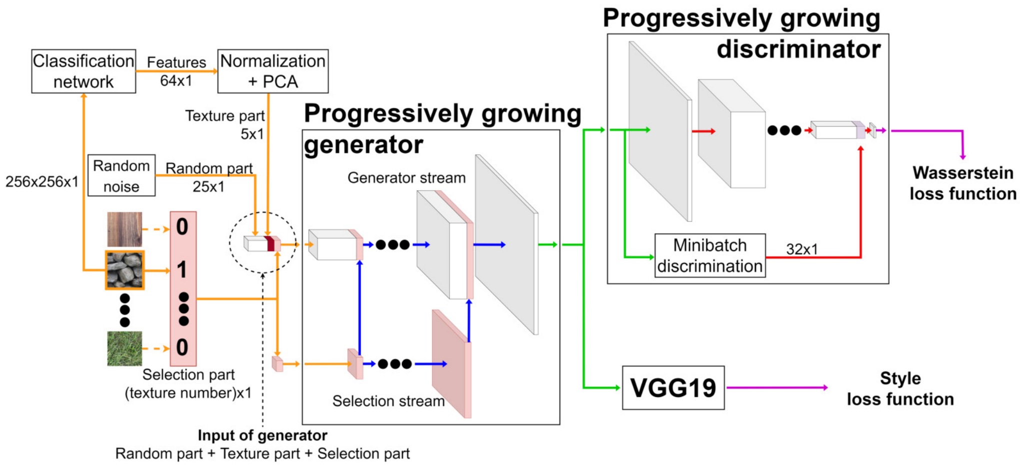

Figure 1: Visualization of the architecture of our solution.

3.2 Generator inputs

As a consequence of two generator streams, the generator requires two inputs. Input for secondary streams is realized as a one-hot encoding vector. This vector is inspired by the solution of Li et al. [14] and encoded as position of a single high bit, that clearly describes which texture should be synthesized. Primary input is inspired by the solution of Bergmann et al. [4]. In contrast with their solution, our input is not constructed as a matrix of vectors but only as one vector. The input vector is concatenated from three parts: random, texture, and selection.

- The random part is sampled from the uniform distribution on the interval [0,1). This part provides variability between generated outputs.
- The selection part is a copy of the input for the secondary stream. One-hot encoding vector is included in primary input to contribute information about selected texture even at the first layers of the generator.
- The texture part is created from an example image of the required texture class. Example image transformed to grey-scale is used as input for our classification network. We took input activations of the last fully connected layer as a feature vector. We apply principal component analysis on this vector to obtain a representation vector of length five numbers. This part of input assists to the selection part with information about selected texture and brings variability to input. To confirm that this vector carries information about the selected texture type, we visualized part of the vector in 3D space. As we can see in example figure 2, every cluster of texture type could be approximately separated.

We selected the classification network and principal components analysis approach for initial experiments because of relatively more straightforward network training than the Siamese neural network for texture similarity.

The intention behind two different information of selected texture was to exactly specify what texture type should be synthesized (selection part) and systematically control differences inside of generated class (texture part) like color, sizes of visual features of content in concrete texture type. Experiments showed that the currently presented architecture and texture part of input does not affect output as expected. However, two texture controlling mechanisms helped learn all presented textures in relative quality. In the case of omitting one of the controlling mechanisms, a significant problem with overall quality or inability to generate some texture types occurred.

We developed a solution that is, thanks to the selection part of input, independently similar to the solution presented by Alanov et al. [1]. However, our approach's main difference is using a pretrained classification network and principal component analysis rather than an encoder to gain information about texture type from example texture. We expect better embedding due to supervised learning of classification network, compared to unsupervised learning of auto-encoder, where is no guarantee that embedded space is flattened enough to position two technically similar texture patches close to each other. We also merged this approach with generator architecture presented by Li et al. [14] to help retain information about the requested texture type during the whole synthesis process. In the end, we changed networks to progressively growing to stabilize training and improve results of non-stationary texture types.

3.3 Discriminator

The initial discriminator used in our solution was inspired by the DCGAN solution [20]. We changed it by adding a sequence of batch normalization, leaky relu activation function, and convolutional layer with kernel size 5, stride 1, and zero padding behind the last convolutional layer. Because of the implementation of the progressive, growing generator, the discriminator also had to be imple-

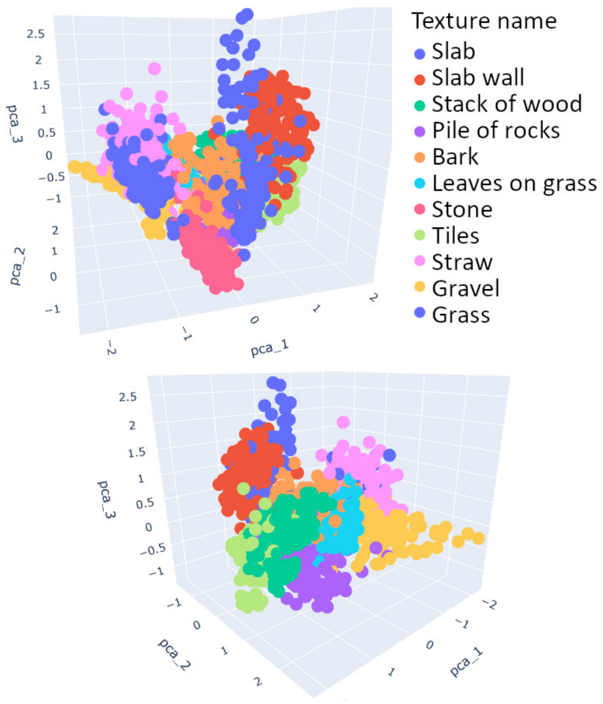

Figure 2: A visualization of the clusters of 3 out of 5 values from the texture part.
Note: Because of multiple texture types, two classes are shown with blue color.

mented as progressively growing. That caused a change of the parameters of the first convolutional layer to kernel size 3, padding, and stride set to 1. A number of the following sequences of convolutional, normalization, and activation layers from the DCGAN solution is based on the current step of progressive growth. We also added a fade-in mechanism to handle adding new layers during progressive growth. After the last convolutional sequence, we added a flatten layer to create a 1D vector. To deal with the problem of mode collapse, we concatenated the final 1D vector of the discriminator with 1D vector from minibatch discrimination [21] to help the network identify a batch of generated images. Minibatch discrimination compares all images in batch and creates a vector that references how similar images are. In case of high similarity, we can assume that images are synthesized and suffer from mode collapse. Based on executed tests, we noticed that minibatch discrimination helps mainly at the first two steps of progressive growth, which correspond to images of size 8×8 and 16×16. The final change was to switch the sigmoid layer to a linear layer to constrain the output vector values to interval $(0-1)$.

3.4 Loss functions

To increase output quality and stability of training, instead of the original loss function used in Goodfellow's proposed GAN solution, we used Wasserstein loss [2] with weight clipping. That changed our discriminator to critic, which does not determine real or fake textures but tells the distance between the distribution of generated outputs and training data. We combined Wasserstein loss with style loss during the training process to speed up the training process and bring a closer visual appearance of synthesized and real textures. To calculate texture loss, we used feature vectors of the first five convolutional layers of the pretrained classification network VGG19. Feature vectors of synthesized and real textures were compared using Gram matrices. Style loss L_{style} is added to Wasserstein loss W for generator loss L_g and could be controlled by the weight factor β. Based on experimenting with the value of β, we found that the ideal value in our setup is 1. In the case of a higher style weight factor, the network learns style relatively fast, but outputs suffer from strong mode collapse and low quality. Formulas to calculate loss functions for generator and discriminator are shown in equations 1 and 2 with L_d as discriminator loss.

$$L_g = -W_{synth} + \beta * L_{style} \tag{1}$$

$$L_d = W_{real} - W_{synth} \tag{2}$$

3.5 Dataset

The dataset used for training our solution consists of eleven different classes of textures. Concretely: slab, slab wall, a stack of wood, a pile of rocks, bark, leaves on grass, stone, tiles, straw, gravel, and grass. Examples of all texture types are shown in figure 3.

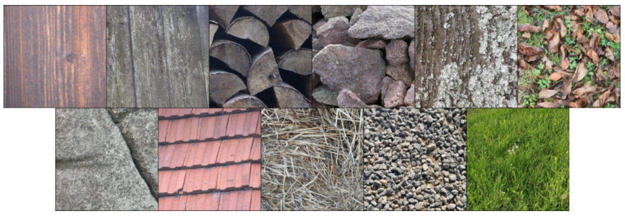

Figure 3: Example patches of 11 texture classes used in the training dataset.

We gathered our own data by capturing close-up photos with textures of real environment objects and adding two freely available textures from the Internet to balance the number of photos in individual classes. Every texture class contains 67 main images. Main images in the dataset are used to create smaller cutouts of various sizes to augment the dataset. Because of differences of the scale of base texel structures in the main images, we scaled various sizes of cutouts to generalize model to the variability of input images. Cutouts positions are selected randomly to get numerous different cutouts of training data. Cutouts for training are then down-scaled to the size of data in mini-batch, which is always smaller than the size of created cutouts.

During the creation of the dataset, our main goal was to select non-stationary homogeneous textures. Because of this condition, we mostly selected natural textures that are separable from their surroundings and do not have any regular pattern. However, in a few cases, we violate our intention to gain information about the behavior of our solution in special cases. These cases are represented by adding texture of tiles that is stationary and texture of leaves on grass that is heterogeneous. We also added two similar textures, slab and slab wall, to determine how well our solution will generate multiple very similar textures.

3.6 Training

During the training of multiclass synthesis, various techniques that affect the alternation of classes could be used. We decided to use a simple cyclic alternation that ensures the change of texture class after every batch. We implemented this alternation as a modulo operation of batch number (count of batches) and a total number of texture classes where the result corresponds to the currently synthesized/learned texture.

It is usual practice to violate the symmetry of learning and train the discriminator more often than the generator. The typical learning ratio between discriminator and generator is 5:1 (the generator is updated every fifth update of the discriminator). This helps the discriminator be trained more than the generator and provides better feedback for generator training. This technique is in our solution also implemented as the modulo of batch number and learning ratio. Because of that, it is necessary to ensure that the generator is trained on every texture class. That could be achieved by setting the learning ratio and the total number of textures to be co-prime. Because of the typical learning ratio (5:1), the easiest way to preserve all conditions is to choose the number of texture classes indivisible by five.

Whereas our solution is progressively growing, we also needed to ensure a mechanism to control when network layers to be trained are added. This issue is resolved using predetermined values assigned to each step of progressive growth, indicating how many epochs each step will be trained. This method, despite its simplicity, allows control over the length of training steps because later steps of the progressive growth need longer training than the first steps with low output resolution.

Yet final training was trained for 1375 epochs unevenly divided into 5 steps from 8×8 to 128×128 pixels (100, 200, 250, 375, 450 epochs). Resolution of every step is double of the previous resolution. Every epoch consisted of 275 batches of 64 images. The training takes about four and half days on a single NVIDIA GeForce RTX 3090 GPU. As optimizers, we used RMSprop with a learning rate set to 5×10^{-5} for the generator and 9×10^{-5} for the discriminator.

4 Evaluation

Considering the problem of non-stationary texture synthesis, one of the most quality accurate ways of evaluation that works even on a low number of generated outputs is empirical evaluation by human eye. This type of evaluation is a qualitative technique based on human observation and feelings of how accurate the texture is or how easily it could be identified as fake. Thanks to this type of evaluation, we were able to estimate the quality of our solution from the very beginning of implementation and lead the architecture to better results.

Based on results from the latest version of our solution, which is described in this paper, and the current version of the dataset, we are able to synthesize 7 out of 11 texture types with tolerable quality. Examples of all 11 texture types with labels are shown in figure 4. Textures that are not considered of sufficient quality (wood, pile of leaves on grass, and tiles) could be easily labeled as fakes at first sight without closer examination. We also noticed that outputs of very similar texture types (slab and slab wall) could not be separated on 100%, and the texture of slabs contains vertical lines that indicate multiple slabs that form wall.

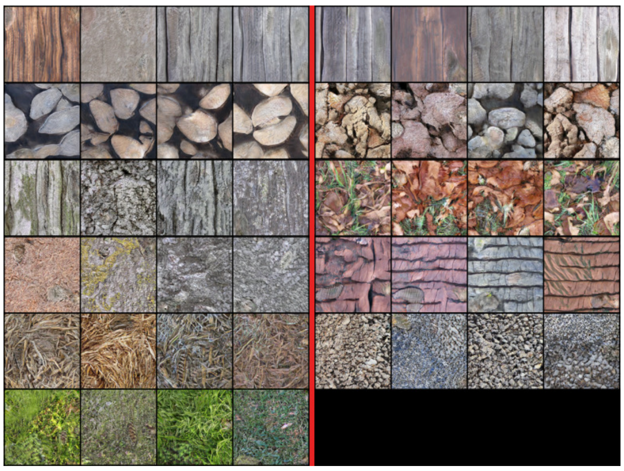

Figure 4: Examples of synthesized textures. Left section from top to bottom: slab, stack of wood, bark, stone, straw, grass. Right section from top to bottom: slab wall, pile of rocks, leaves on grass, tiles, gravel.

Except for the synthesis of concrete learned types of texture, multiple texture controlling mechanisms of input in our solution also allow the possibility of combining texture types on generator input to create new types of textures. Inspired by the work of Li et al. [14] we also tested interpolation of texture controlling parts of input to create transitions between two types of textures. Based on testing of multiple merge operations on texture and selection parts of two types of textures, we found out that the best transition between texture types is obtained by using equation 3 to both texture and selection vectors. A and B are pairs of texture part vectors or selection part vectors. Alfa is the weight of the first vector and is distributed on [0, 1].

$$C = \alpha * A + (1 - \alpha) * B \quad (3)$$

This transition looks good between a pairs of textures that are quite natural, and the probability of their transition in nature is higher than, for example, a transition between grass and tiles. However there are still artifacts that visibly worsen the photorealism of the generated texture. Examples of transitions are shown in figure 5.

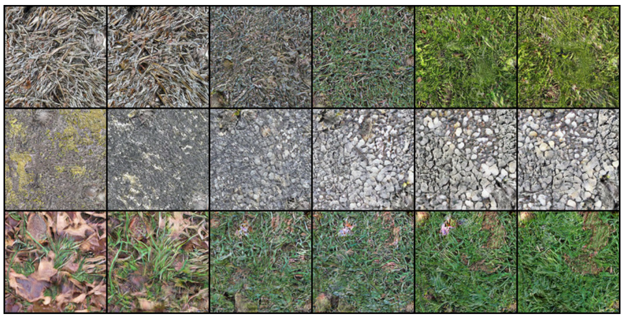

Figure 5: Examples of transition between texture types. From top: Straw to grass, stone to gravel, leaves on the grass to grass.

5 Siamese network experiment

During our research, we experimented with the training of the Siamese neural network as a tool for extracting the texture part of input and learning of similarities between textures. The network output was a vector of length three that was scaled with a min-max scaler to range [-1,1]. As shown in figure 6 network was able to separate texture types into 3D space better than our original approach with classification network and principal component analysis.

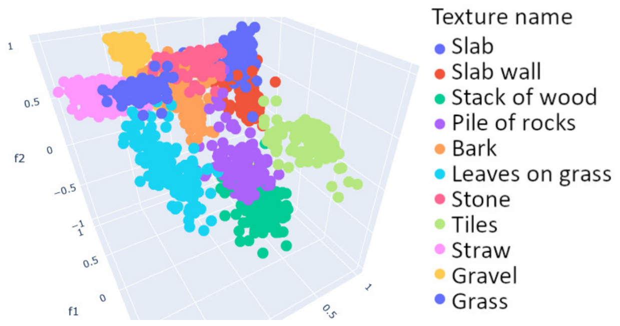

Figure 6: A visualization of the clusters from the texture part obtained from a Siamese neural network.
Note: Because of multiple texture types, two classes are shown with blue color.

As part of the experiment, we modified our architecture and omitted the selection part of the input. Thus we used only texture and random part for primary input and assigned texture part for secondary input. With this approach, we identified that model is able to learn all types of textures compared with a similar setup of our original texture part from the classification network. However, based on evaluation with Fréchet Inception Distance, the overall output quality is lower than the original approach with equal learning iterations and setup.

6 Discussion

Based on the results, we analyzed possible problems and proposed ideas on how our method will be improved in the next iterations. Characteristics of some texture types or features of images used in the training set could explain the low quality compared to other synthesized texture types.

In the case of tiles and leaves on grass textures, these textures do not belong to a specific group of non-stationary homogeneous textures like the rest of the dataset. As the human-created non-natural objects, tiles are strictly stationary because of their standard layout on roofs. Because of training mainly on non-stationary textures, our solution cannot reconstruct periodically repeating textures. The solution's full potential must be confirmed on a dataset containing more stationary textures. Almost every texture type in the used dataset is homogeneous, unlike the texture of leaves on grass that is heterogeneous. Even though generated results of this texture class kept heterogeneity, leaves are not detailed enough. We assume that more extended training of individual progressive growing steps could help with this issue. That will be tested in future iterations of work.

We assume that the problem with the texture of the pile of rocks is caused by internal variation of images in the training set for this class. It means that the dataset contains a mix of rocks images with different sizes and colors of rocks. During training with a subset of the dataset, where the class of rocks contained only one type of rocks, the model was able to synthesize quite satisfying results that could be seen in figure 7.

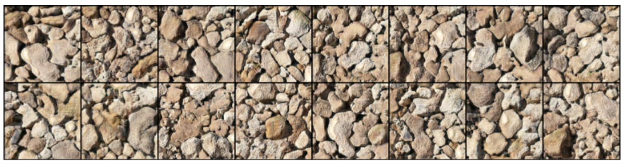

Figure 7: Example of synthesized piles of rocks from a model that was trained on a dataset where the class of piles of rocks contained only one type of rock.

7 Conclusion and Future work

In this paper, we presented a multiclass texture synthesis model based on generative adversarial networks. Our approach is tuned to synthesize non-stationary textures that are problematically synthesized by traditional approaches. Parts of existing solutions inspire the presented model and

GAN mechanics to obtain user controllability, preservation of information about selected texture along the entire length of the generator, training stability, and reduction of mode collapse. To the input of the generator, we added information about the selected class from an example texture. However, this turned out to be unintentionally similar to the technique proposed by Alanov et al. Nevertheless, the method of obtaining information from example texture is different.

We evaluated the results of our approach by the qualitative technique of human observation of synthesized textures. Based on this evaluation, we found that our solution could synthesize 7 out of 11 training texture types with sufficient quality. In the case of the majority of insufficient textures, we analyzed possible reasons for poor quality that led to a plan for future work.

In the subsequent phases of our work, we plan to focus on improving our solution's output quality and controllability. Our main idea is to improve the texture part of generator input by continuing with experiments with Siamese neural network that is able to learn representations of textures and put textures from the same class closer and textures from different classes far apart. An interesting idea is also to increase the dimensional space of the texture part and force the Siamese network to distribute output that it will be possible to transit between all types of textures.

We also want to improve the evaluation of our approach by using quantitative metrics, not just qualitative. In the case of qualitative human observation, we intend to create a user questionnaire to evaluate our solution on the general public.

References

[1] Aibek Alanov, Max Kochurov, Denis Volkhonskiy, Daniil Yashkov, Evgeny Burnaev, and Dmitry Vetrov. User-controllable multi-texture synthesis with generative adversarial networks. pages 214–221, 01 2020.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. 01 2017.

[3] Michael Ashikhmin. Synthesizing natural textures. *Symposium on Interactive 3D Graphics*, 12 2000.

[4] Urs Bergmann, Nikolay Jetchev, and Roland Vollgraf. Learning texture manifolds with the periodic spatial GAN. *34th International Conference on Machine Learning, ICML 2017*, 1:722–730, 2017.

[5] Rohan Chandra, Sachin Grover, Kyungjun Lee, Moustafa Meshry, and Ahmed Taha. Texture synthesis with recurrent variational auto-encoder. 12 2017.

[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 vol.2, 1999.

[7] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. *CoRR*, abs/1505.07376, 2015.

[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 3(January):2672–2680, 2014.

[9] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein GANs. *Advances in Neural Information Processing Systems*, 2017-Decem:5768–5778, 2017.

[10] Filip J. Haindl M. *Visual Texture*, chapter Motivation. Advances in Computer Vision and Pattern Recognition. Springer, 2013.

[11] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2017.

[12] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[13] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003.

[14] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming Hsuan Yang. Diversified texture synthesis with feed-forward networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:266–274, 2017.

[15] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.*, 20:127–150, 07 2001.

[16] Randi Martin and James Pomerantz. Visual discrimination of texture. *Perception & Psychophysics*, 24:420–428, 09 1978.

[17] Mehran Pesteie, Purang Abolmaesumi, and Robert N. Rohling. Adaptive Augmentation of Medical Data Using Independently Conditional Variational Auto-Encoders. *IEEE Transactions on Medical Imaging*, 38(12):2807–2820, 2019.

[18] Javier Portilla and Eero Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40, 10 2000.

[19] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *Proceedings of ACM SIGGRAPH 2000*, pages 465–470, July 2000.

[20] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. 11 2016.

[21] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[22] Seunghyup Shin, Tomoyuki Nishita, and Sung Yong Shin. On pixel-based texture synthesis by non-parametric sampling. *Computers and Graphics (Pergamon)*, 30(5):767–778, 2006.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[24] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. *Computer Graphics (Proceedings of SIGGRAPH'00)*, 34, 05 2000.

[25] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *ACM Transactions on Graphics*, 37(4), 2018.

[26] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. pages 2242–2251, 10 2017.