Neural Texture Synthesis

Shaoyang Cui
Yuanpei College, Peking University
2100012521@stu.pku.edu.cn

Siyuan Yin
Yuanpei College, Peking University
2100017768@stu.pku.edu.cn

Xiaohui Zhang
School of Mathematical Science, Peking University
2200010821@stu.pku.edu.cn

Abstract

The realm of texture generation has undergone notable advancements through the integration of Convolutional Neural Networks (CNNs) and Markov Random Field (MRF) methodologies. This project aims to develop a general neural network model, based on the VGG-19 architecture, capable of accurately describing and synthesizing a diverse array of textures. We carefully examine the effects of various pooling strategies, rescaling methods, and optimizers, and present a comprehensive series of experiments to illustrate these impacts.[1] *Furthermore, our investigation delves into the GCD-loss, highlighting its efficacy and stability in enhancing texture generation capabilities.*

1. Introduction

Texture research plays an important role in realism in computer graphics, material recognition and enhancing machine perception. In computer graphics, realistic texturing is crucial for creating lifelike images and animations[12]. In medical imaging, texture analysis can assist in identifying different tissue types and diagnosing diseases[6]. In object recognition, scene understanding and robotic navigation, understanding the texture of objects can provide critical information about their nature and condition[14]. Texture analysis and synthesis are active research areas, pushing the boundaries of how computers interpret and generate visual content. Understanding texture is an essential part of understanding human vision.

As defined in Julesz Ensemble[15], texture can be described as a set of features H, so all the image I of texture $\hat{H}$ can be defined as:

$$\{I: \quad h_i(I) = \hat{h_i} \quad \forall h_i \in H\} \tag{1}$$

[1]Code is available at https://github.com/Wanderings0/Neural-Texture-Synthesis

Then we offer the definition of texture analysis and synthesis respectively:

- the goal of **texture analysis** is to learn a probability distribution $p(I)$ based on given texture $\{I_1, I_2, ..., I_n\}$, which satisfy:

$$E_p[h_i] = \hat{h_i} \quad \forall \hat{h_i} \in \hat{H}, \quad h_i \in H \tag{2}$$

- **texture synthesis** means sample a texture image I based on the probability distribution $p(I)$: $I \sim p(I)$

There are primarily two categories of promising models used in texture analysis and synthesis: discriminative models and generative models. Generative models incorporate a hidden variable, denoted as z, and reconstruct the image through the probability function $p(I|z)$. This approach is exemplified by technologies such as Auto-encoders[11], Generative Adversarial Networks (GANs)[4], and Denoising Diffusion Probabilistic Models(DDPMs)[5]. On the other hand, Discriminative models focus on reconstructing the image based on sufficient statistics, represented by the function $p(I|\partial I; \theta)$. Methods of this approach always utilize Markov random field(MRF)[1] for sampling because of the intractable p. Based on MRF, Gaussian Markov Random Fields (GMRF)[7][8] and the Julsez Ensemble model & FRAME model[18] [15] are effective methods for texture learning. Extracting statistics properly is a key question for discriminative models. And with the development of deep learning, there are more powerful way to extract sufficient statistics like convolutional neural network(CNN)[9, 13], the CNN greatly strengthens those probabilistic models[6].

We develop a general neural network model, based on the VGG-19 architecture, capable of accurately describing and synthesizing a diverse array of textures. We make some modifications on model architecture and carefully examine the effects of various pooling strategies, rescaling methods, and optimizers. We use gram matrix and mean square error(MSE) as loss function and use optimizer to optimize

synthesized image. We present a comprehensive series of experiments to illustrate these impacts. Furthermore, our investigation delves into the GCD-loss, as proposed by Zhou et al.(2023)[16], highlighting its efficacy and stability in enhancing texture generation capabilities in the additional discussion part.

2. Related Work

MRF: To capture the special statistics of images, we usually define an energy function U of image I and build an exponential form probability distribution $p(I) = exp\{-U(I)/T\}$that satisfies Markov property, which is, on the image space, $p(I_s|I_{\bar{s}}) = p(I_s|I_{\partial s})$. And the Markov Random Fields (MRFs)[1, 2], which describes a set of variables having a Markov property described by an undirected graph, is widely employed in texture learning and generating tasks for its ability to sample from the intractable probability distribution $p(I_s|I_{\partial s})$[7][8][18] [15].

VGG: As defined in Eq 1, building up a proper energy function $U(I)$ requires us to extract sufficient statistics from image I that correspond to the ensemble $\hat{H}$. Although getting used to manually design feature h_i and $U(I)$, with the development of Deep Learning and Convolutional Neuron Network(CNN)[9], researchers prefer to take the advantage of pretrained neural network to get more perfect feature maps[6]. VGG network[13], as a famous neural network in image processing, can feel equal to this work with pretrained parameters. Activations after each layer of the network VGG can be viewed as feature maps which will be used to calculate gram matrix and even optimizing the image later.

CNNMRF: CNNMRF, proposed by Li et al[10], combines MRF and neural networks for style transfer and texture synthesis. It uses discriminatively trained deep Convolutional Neural Networks (dCNNs) to get a feature pyramid, and take advantage of MRF theory to generate texture. Unlike standard MRF method, this combined system can both match and adapt local features with considerable variability, thus performing better than the original method[3].

3. Method

3.1. Model Modifications

We adapted the VGG-19 based neural texture synthesis model by removing its fully-connected layers. Additionally, we incorporated two significant modifications for texture synthesis.

- Substituting the max-pooling operation with average pooling.
- Adjusting the weights to normalize the mean activation of each filter across images and positions to a value of one.

Average pooling tends to preserve more representative feature whereas max-pooling focuses on the most prominent features, potentially neglecting subtle but important information. Average pooling can result in smoother, more continuous representations of textures, which is crucial in texture synthesis tasks.

By normalizing the mean activation of each filter to one, the model ensures a more balanced and uniform response across all filters. This balance helps in maintaining the integrity of various textural features, ensuring that no single feature disproportionately influences the synthesized texture. This can lead to more accurate and diverse texture generation. We use the ground truth image to rescale the weights to normalise the mean activation of each layer.

3.2. Texture Analysis and Synthesis

In Neural Texture Synthesis, we need to extract some statistical invariant. Here we use the Gram Matrix to measure the style of an image. It can represent the relationship between different channels of a feature map and show some summary statistic that discards the spatial information. In terms of its calculation, The Gram Matrix of a feature map is defined as the inner product of the vectorized feature map with itself. When comparing in the field of mathematics, Gram Matrix is similar to the covariance matrix of the feature map.

Covariance between two random variables is as follows:

$$\begin{aligned}\mathrm{Cov}(X,Y) &= \mathbb{E}[(X-\mathbb{E}[X])(Y-\mathbb{E}[Y])] \\ &= \mathbb{E}[XY]-\mathbb{E}[X]\mathbb{E}[Y]\end{aligned}$$

If $X = Y$, the covariance degrades into the variance of X.

When X is a vector, the covariance matrix is defined as:

$$\begin{aligned}\mathrm{Cov}(X) &= \mathbb{E}[(X-\mathbb{E}[X])(X-\mathbb{E}[X])^T] \\ &= \mathbb{E}[XX^T]-\mathbb{E}[X]\mathbb{E}[X]^T\end{aligned}$$

Now we turn to the Gram Matrix of a feature map F with C channels and $H \times W$ spatial size. The Gram Matrix is defined as:

$$G^l_{ij} = \sum_{k=1}^{C} F^l_{ik} F^l_{jk}$$

where G^l_{ij} is the activation of the i-th channel and the j-th channel of the l-th layer.

After the Gram Matrix is calculated, we can use it to measure the style of an image. Given a white noise image X and a texture image Y, we can calculate the Gram Matrix of the feature map of X and Y respectively. Then we can use the mean squared error between the Gram Matrix of X and Y to measure the loss. The loss is defined as:

$$\mathcal{L}_{style} = \sum w_l E_l$$

$$\text{where } E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - A^l_{ij})^2$$

Because the loss E is calculated by the Gram Matrix getting from the feature maps, which is directly related to the original pixel values of the image, of course we can use the gradient descent method to minimize the style loss. Here we use the optimizer to update the pixel values of the white noise image. When iterating, the white noise image will gradually get similar texture with the texture image.

3.3. L-BFGS Algorithm

L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) algorithm is an optimization algorithm in the family of quasi-Newton methods. It maintains a history of recent position and gradient vectors to implicitly represent the approximation of the inverse Hessian, which makes it particularly suitable for optimization problems with a large number of variables. That's exactly why we utilize it. L-BFGS employs a line search algorithm to find the step size that will be taken along the search direction. Under certain conditions, it can achieve superlinear convergence, faster than gradient descent. In a word, L-BFGS balances between memory usage and convergence speed, which makes it an attractive choice compared to gradient descent and full-memory BFGS algorithms. There are some simplified mathematical formulations to show the principle. The denotation is shown in Table 1.

Denotation	Explanation
$f(x)$	the objective function to be minimized
s_k	$s_k = x_{k+1} - x_k$
y_k	$y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
ρ_k	$\rho_k = \frac{1}{y_k^T s_k}$
p_k	$p_k = -B_k^{-1} \nabla f(x_k)$ is an approximation to the inverse Hessian matrix(B_k^{-1})

Table 1: Denotation

To obtain the search direction p_k , L-BFGS uses a two-loop recursion to apply these updates to the gradient: First loop computes a sequence of vectors q_i starting with $q_m = \nabla f(x_k)$ and updating q_i based on s_i and y_i. Second loop builds the search direction p_k from the q_i. The algorithm is shown in Algo 1.

3.4. Data

In the subsequent experiments, including the additional discussion section, we utilized texture data selected from [17]. We chose some distinctive textures from this dataset, as shown in Figure 1, to serve as the learning subjects for our model in the following experiments. Our CNNMRF model will generate textures based on these examples.

Algorithm 1 L-BFGS

$q \leftarrow$ gradient
for $i \leftarrow k$ **down to** $k - m$ **do**
    $\alpha[i] \leftarrow \rho[i] \times s[i]' \times q$
    $q \leftarrow q - \alpha[i] \times y[i]$
end for
$r \leftarrow H_k^0 \times q$
for $i \leftarrow k - m$ **to** k **do**
    $\beta \leftarrow \rho[i] \times y[i]' \times r$
    $r \leftarrow r + s[i] \times (\alpha[i] - \beta)$
end for
$p_k \leftarrow -r$

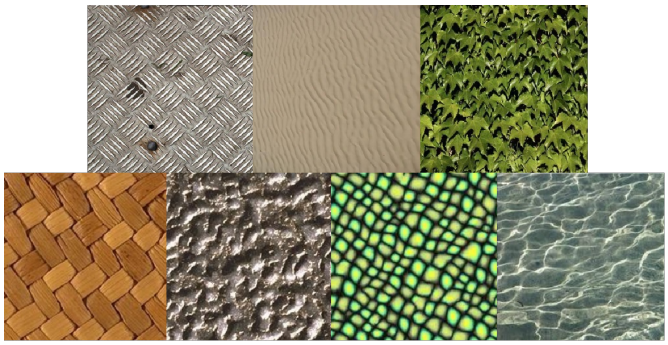


Figure 1: Texture images

4. Experiments

In this section, we offer a detailed discussion of our experiments, we emphasize that all those experiments are carried out with configurations shown in Table 3.

4.1. Effect of feature selection(Constraints on feature scales)

The multi-scale network structure of VGG-19 provides us with various options for feature selection. To explore more efficient constraints on feature scales, we conducted experiments with features from different scales of the network. Specifically, during the texture generation process, we used feature maps from different scales of the VGG-19 to calculate loss. These experiments were carried out both on the original VGG network and on the network after rescaling and pooling replacements. And the results of these experiments are illustrated respectively in Figure 2, 3. Check table 2 for the corresponding feature of each scale.

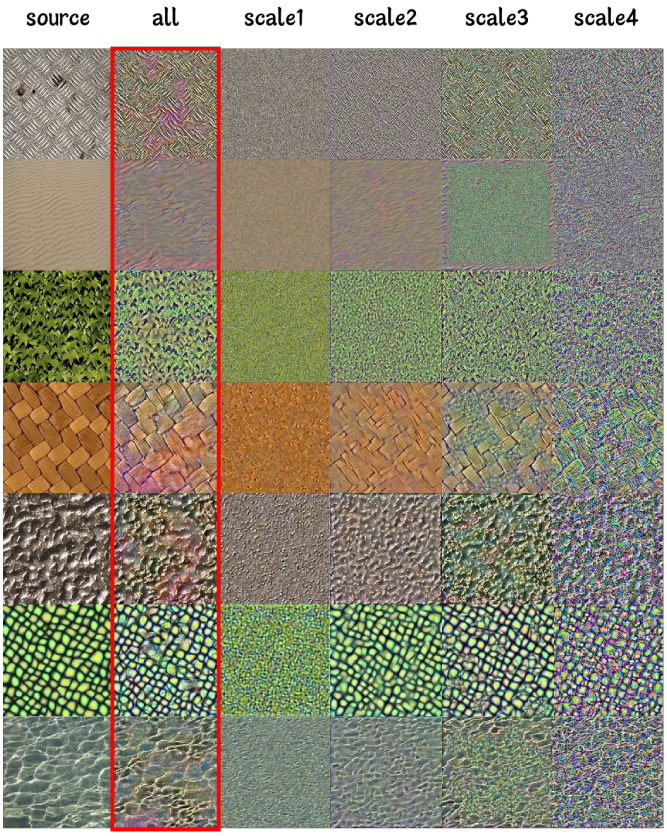


Figure 2: Generating results based on **origin pre-trained VGG-19**. Column Scale i shows the generating results using only the first activation in scale i. 'all' indicates generate image with all features from all scales.

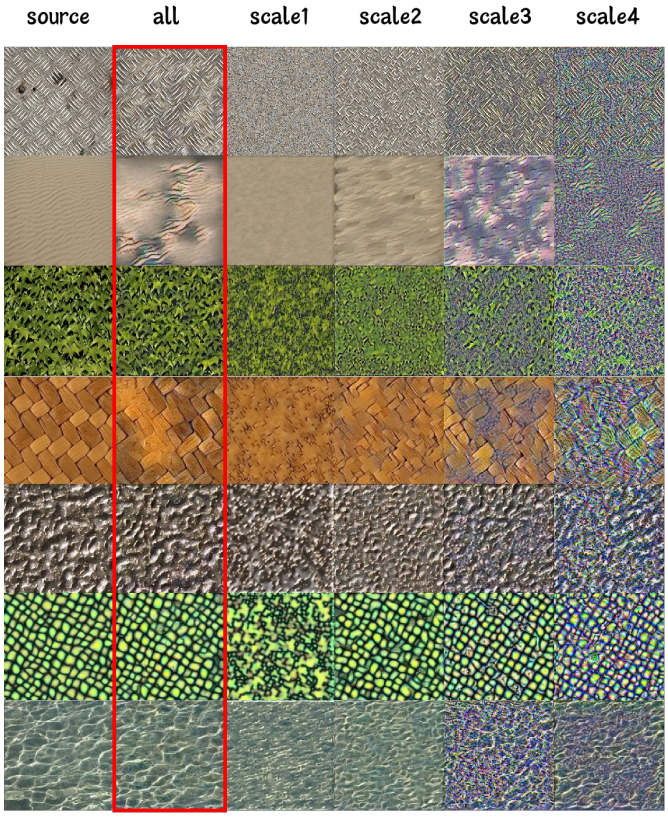


Figure 3: Generating results based on **rescaled VGG-19** with **average pooling layers**. Column Scale i shows the generating results using only the first activation in scale i. 'all' indicates generate image with all features from all scales.

Scale	Explanation
scale 1	features before pooling layer 1
scale 2	features after pooling layer 1
scale 3	features after pooling layer 2
scale 4	features after pooling layer 3

Table 2: Scale Layer correspondence

From the results, we can observe that as the scale deepens, the network tends to extract coarser-grained features such as edges and textures. In contrast, shallower layers of the network are more adept at capturing features like colors and basic textures. This observation aligns with our understanding of deep convolutional networks.

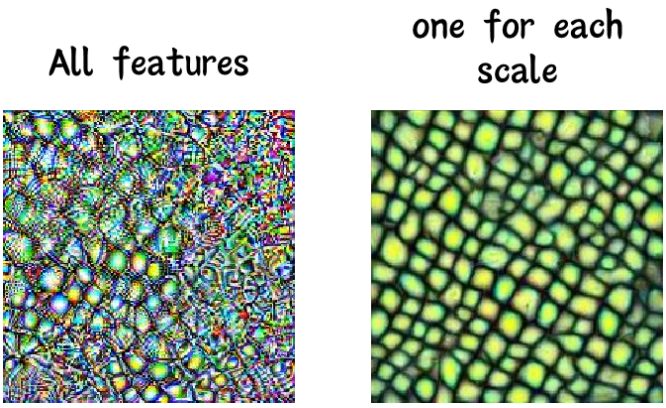
Additionally, we discovered that the results generated using only one feature from each scale were much better than those obtained using all features of that scale, which enable us greatly reduced the time consumption while getting a better result(Shown in Figrue 4).

4.2. Comparisons on rescaling and pooling method

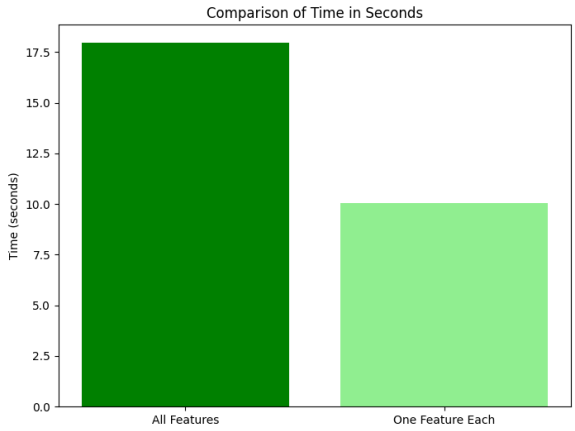
As described in section 3.1, we carried out experiments to see their effects(shown in Figure 5). We found the for both origin and rescaled model, average pooling layer leads to a better results. Meanwhile, the rescaled model performed better on images with coarser-grained features, such as those depicting metal or leaves. However, when dealing with textures having very fine grains, like sand in a desert, its performance was not as good as the original model.

4.3. LBFGS

As described in Section 3.3, we implemented the optimizer LBFGS and carried out contrast experiments to compare it with Adam. The results are shown in Figure 6. We found that despite Adam's ability to rapidly converge the loss to very low values, from the perspective of image generation quality, LBFGS is capable of achieving results close to the ground truth in about 100 epochs. In contrast, Adam requires around 400 epochs to reach a similar standard. This indicates that LBFGS is more effective for image generation



(a) Results



(b) Time cost

Figure 4: (a)The results images with all/one feature(s) of each scale. (b)Their time costs.

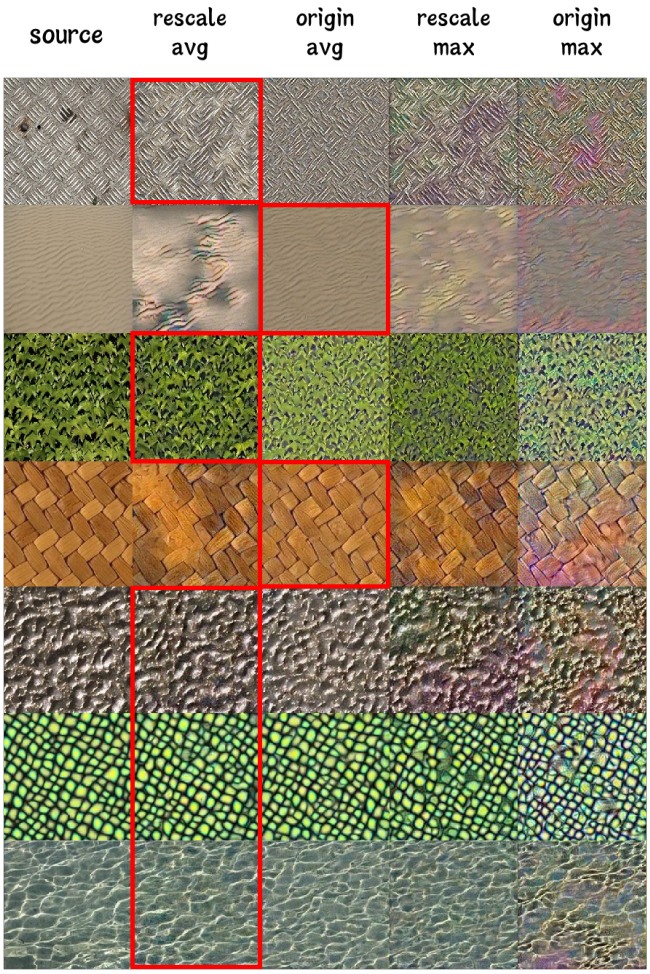


Figure 5: The generating results of: rescaled model with average pooling, rescaled model with max pooling, origin model with average pooling, origin model with max pooling.

tasks.

5. Additional Discussion: Guided Correspondence Distance

5.1. Mathematical foundations

In this section, we will explain what [16] did to leverage CNNMRF with guidance and why it works.

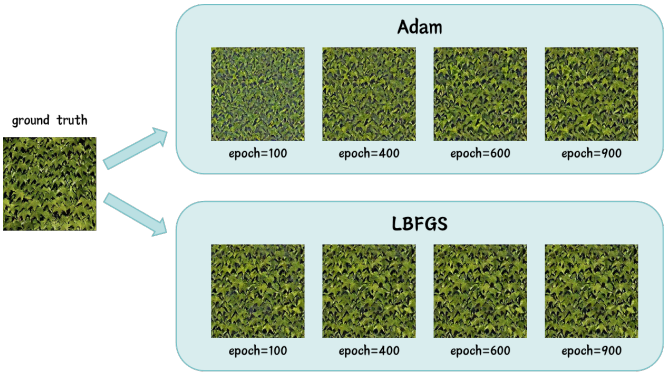
1.The CNNMRF: Use a CNN to extract the feature from both input/source image I_s and output/target image I_t for calculating the energy function E of a Markov Random field. Then sample from the MRF based on E

For the MRF implemented here, we note it as:

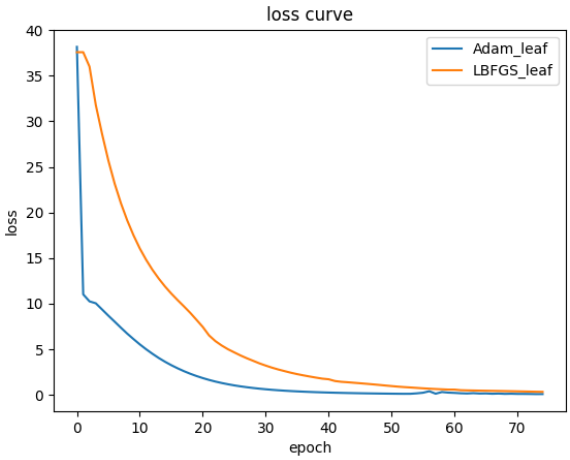
$$E_{MRF}(I_s, I_t) = \sum_{t_i \in T} p(t_i, NN(i)), \text{where}$$

- T, S respectively represent the collection of overlapping patches: $T = \{t_i\}$ and $S = \{s_i\}$
- $NN(i)$ indicates the nearest neighbour of t_i in S: $NN(i) \in S$.
- $p(.,.)$ is the similarity function, generally the sum of square color distance in regular MRF models, but here we modified it for better performances.

For classical MRF models, an Estimation-Maximization like method will be utilized to solve the object, however, Zhou et.al design a special similarity function, convert it to a loss function and minimize it with back-propagation, which we will discuss later in details.



(a) Results



(b) Loss Curve

Figure 6: (a)Generating results of image 'leaf' using Adam and LBFGS optimizer (b)The loss curves of Adam and LBFGS

5.1.1 Guided Correspondence Distance

TThe definition of GCD includes three crucial parts: (1) distance between the features from source image and target image extracted by VGG, (2) distance of features from image and guidance, (3) the penalty of occurrence. Here, we first offer the formula of distance between patch t_i, s_j, noted as d_{ij}.

$$d_{ij} = d_{ij}^{VGG} + \lambda_{GC} \times d_{ij}^{GC} + \lambda_{occ} \times d_j^{occ}, \text{where}$$

- d_{ij}^{VGG} is the cosine distance between neural patches $F_t(i)$ and $F_s(j)$
- d_{ij}^{GC} is the distance between guidance patches $G_t(i)$ and $G_s(j)$
- d_j^{occ} is the occurrence penalty.
- $\lambda_{GC}, \lambda_{occ}$ are parameters.

And the $\{F_t^l(i), G_t^l(i)\}$ $\{F_s^l(j), G_s^l(j)\}$ represent the feature of patch t_i, s_j on layer l of VGG net and guidance.

Then, we just simply offer the detailed definition of d_{ij}^{VGG}, d_{ij}^{GC} and d_j^{occ}.

The distance of VGG feature can be calculated as:

$$d_{ij}^{VGG} = 1 - \frac{(F_t(i) - \mu_s) \cdot (F_s(j) - \mu_s)}{||F_t(i) - \mu_s||_2 \cdot ||F_s(j) - \mu_s||_2}$$

which is exactly the definition of cosine distance and $\mu_s = \frac{1}{n_s} \sum_j F_s(j)$

The occurrence penalty can be calculated as:

$$d_j^{occ} = \frac{\Omega_j}{\omega},$$

where $\Omega_j = |\{s_j = NN(t_i), \forall i\}|$ and $\omega = \frac{n_t}{ns}$ plays as a normalization term.

The definition of Guidance depends on the type of guidance, which can be **Annotation, Progression and orientation**: For Annotation control, we calculate d_{ij}^{GC} as:

$$d_{ij}^{GC} = \frac{1}{k * k} \sum_{x=0}^{k*k} ||G_t(j)(x) - G_s(i)(x)||_0,$$

where k is the patch size, $G_s(i)(x), G_t(j)(x)$ denote the corresponding pixels in the patch and annotation map.

And For progression and orientation control, the d_{ij}^{GC} is calculated respectively as:

$$d_{ij}^{GC} = \frac{1}{k * k} \sum_{x=0}^{k*k} ||G_t(j)(x) - G_s(i)(x)||_2^2,$$

and

$$d_{ij}^{GC} = \frac{1}{k * k} \sum_{x=0}^{k*k} (1 - \frac{|G_t(j)(x) \cdot G_s(i)(x)|}{|G_t(j)(x)| \cdot |G_s(i)(x)|})$$

Now we can calculate the distance considering both texture features, guidance and occurrence, we shift it to a contextual similarity, which requires a target sample to be significantly closer to its nearest neighbor than other patch samples.

$$w_{ij} = exp(\frac{1 - d_{ij}/(min_k d_{ik} + \epsilon)}{h}),$$

The ϵ is set to 1e-5 to prevent division by zero and h here is a hyperparameter. Then we normalize w_{ij} to get the contextual similarity CX_{ij}:

$$CX_{ij} = w_{ij} / \sum_k w_{ik}$$

Finally, based on the energy function of MRF, based on our definition of similarity $p(t_i, NN(i)) = -\frac{1}{n_t} log(CX_{i,NN(i)})$, we have:

$$L_{GC}(I_t, I_s) = \frac{1}{n_t} \sum_i -log(CX_{i,NN(i)})$$

This loss function can be used in both texture optimization and generative model training:

1. For texture minimization, we minimize L_{GC} by back-propagation to modify output pixels.
2. For generative model training, we update network weights based on that error.

5.2. Experiments

Unfortunately, we did not manage to independently implement the optimization mentioned in the article. However, to ensure the completeness of our discussion, we used the code provided in the corresponding GitHub(https://github.com/EliotChenKJ/Guided-Correspondence-Loss.git) repository of the paper. We conducted experiments for different control types. Next, we will introduce the experimental parameters and present the results of these experiments.

5.2.1 No control

First of all, we just remove all those controls and try the method on a simple texture synthesis task, the results are shown in Figure 7.

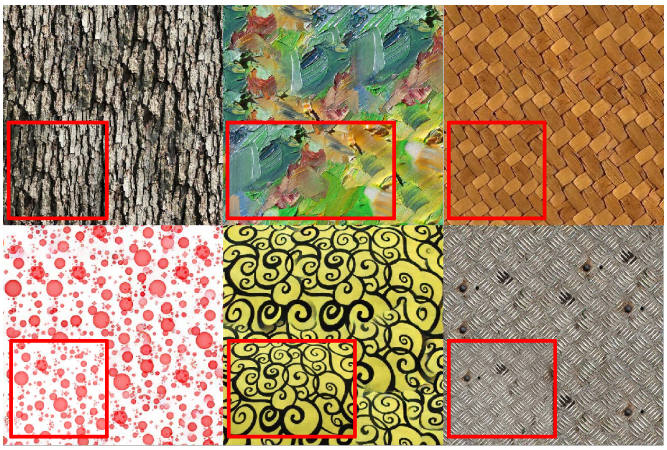


Figure 7: Texture Synthesis without any control. We use red boxes to demonstrate the source texture.

5.2.2 Progression Control

First, we set $\lambda_{orientation} = 0$ and only added progression control, for a comprehensive discussion, we tried all eight kinds of different given by (Zhou et.al) and select three different texture as background, the synthesis results are shown in Figure 8. For configurations, we set $\lambda_{progression} = 50$ and $\lambda_{occ} = 0.05$.

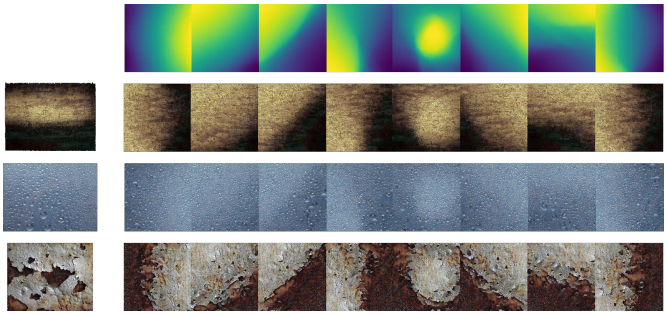


Figure 8: Experimental results under different progression controls

5.2.3 Orientaion contol

We also set $\lambda_{progression=0}$ to test the circumstance with orientation controls only. We have tested all four given controls on bamboo and tree rings with $\lambda_{orientation} = 5, \lambda_{occ} = 0.05$, the results are shown in Figure 9.

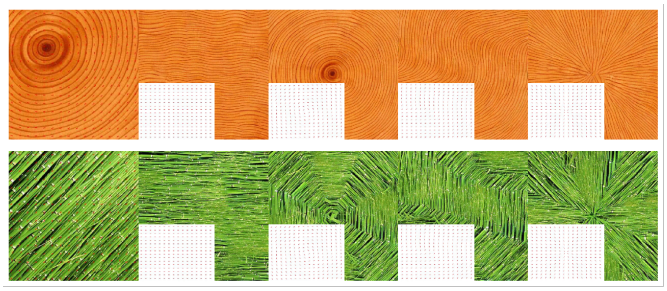


Figure 9: Experimental results under different orientation controls

5.2.4 Both

Also, we test the performance of GCD loss for MRF under both progression and orientation control with $\lambda_{progression} = 10, \lambda_{orientation} = 1, \lambda_{occ} = 0.05$. The results can be seen in Figrue 10

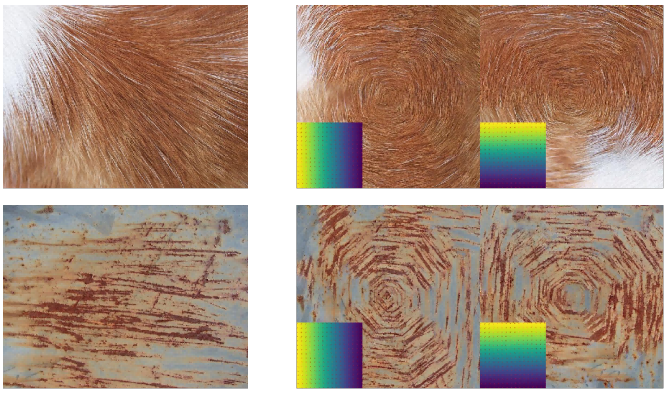


Figure 10: Experimental results under both progression and orientation control

5.2.5 Label

Another application of the method is texture synthesis based on labels, as shown in Figure 11, the leftmost column shows the textures and their corresponding progression(label), and our experiments will generate images with those textures under our given label. In this section, we didn't use any orientation guidance and set a comparatively lower $\lambda_{prog} = 10$ compared with the Progression Only experiment because of the more accurate progression labels.

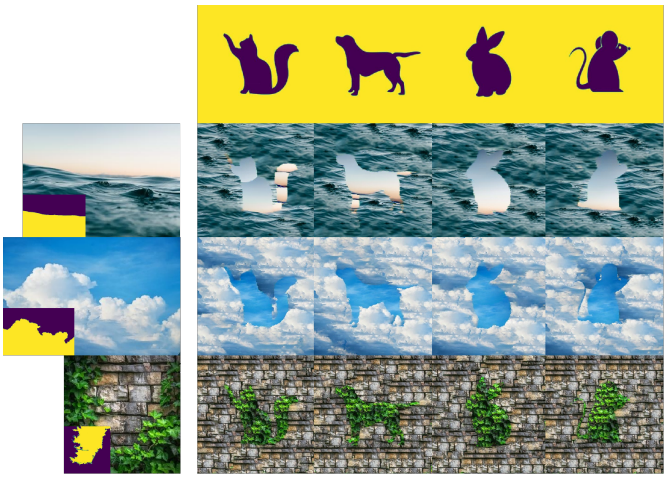


Figure 11: Experimental result under different label controls: [cat, dog, rabbit, mouse]

5.3. Our own control

In this section, instead of regular orientation guidance like "X" or "1", we designed a sketch of face and make it our orientation guidance, Figure 12 shows the results of orientation control only experiments under the same configurations of Section 5.2.3.

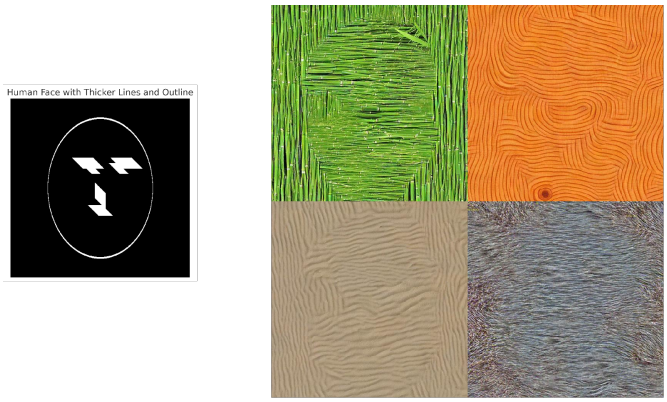


Figure 12: A special orientation guidance of human face sketch

And also, we use both progression and orientation control for a try, the results are shown in Figure 13

Figure 13: Face sketch for both orientation and progression control

6. Conclusion

After a comprehensive series of experiments to illustrate these impacts, we have found that modifying the maxpool in the VGG architecture to avgpool, along with normalizing the mean activation of each filter to one, results in better generation outcomes. Average pooling can result in smoother, more continuous representations of textures, which is crucial in texture synthesis tasks. By normalizing the mean activation of each filter to one, the model ensures a more balanced and uniform response across all filters. This balance can lead to more accurate and diverse texture generation. We use the ground truth image to rescale the weights to normalise the mean activation of each layer.Regarding the choice of optimizer, Adam converges faster than LBFGS. Furthermore, our investigation delves into the GCD-loss, highlighting its efficacy and stability in enhancing texture generation capabilities in the additional

discussion part.

7. Appendix

Our codes are available on **Github**, please check: https://github.com/Wanderings0/Neural-Texture-Synthesis.git

learning rate	0.1
epoch	1000
optimizer	Adam(default)

Table 3: Configuration: CNNMRF

References

[1] Julian Besag. Spatial Interaction and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):192–225, 12 2018.

[2] George R. Cross and Anil K. Jain. Markov random field texture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-5(1):25–39, 1983.

[3] 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.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[5] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020.

[6] Faeze Kiani. Texture features in medical image analysis: a survey, 2022.

[7] S. Krishnamachari and R. Chellappa. Gmrf models and wavelet decomposition for texture segmentation. In *Proceedings., International Conference on Image Processing*, volume 3, pages 568–571 vol.3, 1995.

[8] S. Krishnamachari and R. Chellappa. Multiresolution gmrf models for texture segmentation. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2407–2410 vol.4, 1995.

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[10] Chuan Li and Michael Wand. Combining markov random fields and convolutional neural networks for image synthesis, 2016.

[11] Umberto Michelucci. An introduction to autoencoders, 2022.

[12] Pedro J. Pardo, María Isabel Suero, and Ángel Luis Pérez. Correlation between perception of color, shadows, and surface textures and the realism of a scene in virtual reality. *J. Opt. Soc. Am. A*, 35(4):B130–B135, Apr 2018.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[14] Sachinkumar Veerashetty. Attention-guided deep learning texture feature for object recognition applications. *Engineering Proceedings*, 59(1), 2023.

[15] Ying Nian Wu, Song Chun Zhu, and Xiuwen Liu. Equivalence of julesz ensembles and frame models. *Int. J. Comput. Vision*, 38(3):247–265, jul 2000.

[16] Yang Zhou, Kaijian Chen, Rongjun Xiao, and Hui Huang. Neural texture synthesis with guided correspondence. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18095–18104, 2023.

[17] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *CoRR*, abs/1805.04487, 2018.

[18] Song-Chun Zhu, Ying Nian Wu, and David Mumford. Frame: filters, random fields, and minimax entropy towards a unified theory for texture modeling. *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 686–693, 1996.