university of groningen / faculty of science and engineering

UNIVERSITY OF GRONINGEN

MASTER'S RESEARCH INTERNSHIP

Extensions to Neural Texture Synthesis

Author:
Lonneke PULLES

Supervisors:
Dr. Cara TURSUN
Prof. Jiří KOSINKA

A Research Internship Report submitted in fulfillment of the requirements
for the Master's Degree in Computing Science

in the

Faculty of Science and Engineering

Department of Computing Science

January 8, 2024

Contents

List of Figures

List of Tables

1 Introduction

Natural textures are present all around us and humans are uniquely suited to both recognise and create them. Although a natural texture's pattern is not repeated in an exact or geometric way, the similarity in its spatial recurrences can be detected by a human observer at a moment's glance. Computers, on the other hand, still have more difficulty in performing this task and are not yet at a human level of performance.

1.1 Texture classification

Textures are images that consist of repeating elements, which can be subject to some form of randomization in characteristics such as location and orientation [19]. They can be classified in a spectrum ranging from regular textures (e.g. checkerboards), to near-regular textures (e.g. a brick wall), irregular textures (e.g. a box full of both oranges and lemons), near-stochastic textures (e.g. fire and clouds) and stochastic textures (e.g. stucco) [15]. Figure 1.1 shows examples of textures arranged according to this spectrum. Because of its repetitive nature, a texture can be compressed into a texture representation. This representation can take various forms: the statistical parameters of a stochastic texture, a singular patch of a regular texture, or other. The representations of textures in the middle of the spectrum are less straightforward and require more complex methods.

FIGURE 1.1: A texture spectrum arranged by regularity, from regular to stochastic. Adopted from Lin et al. [15].

1.2 Texture synthesis methods

Texture synthesis is the process of generating a texture via computational methods based on a target texture. Existing methods of texture synthesis can be divided into two overarching categories: non-parametric and parametric. An example of a non-parametric technique is the method by Efros et al. [5], which grows a texture pixel-by-pixel based on the conditional probability of a pixel given its neighbourhood pixels. The estimation of this conditional probability is derived from the target texture, assuming a Markov random field model. Notably, this non-parametric method focuses on image synthesis, and less on analysis via the image's representation. After all, the method does not operate on a compressed representation of the image, but on the original image itself. Parametric techniques, though, have a dual purpose: both their texture analysis and synthesis methods are of interest. Whereas non-parametric techniques do not extract texture characteristics, parametric techniques represent a texture with parameters of the texture, such as features or statistical properties.

Non-parametric methods of texture synthesis yield better results on regular textures, whereas non-parametric techniques are more suitable for irregular textures. Moreover, parametric synthesis techniques generally have a lower computational cost than non-parametric algorithms [1].

The amount of information that a texture representation of a parametric algorithm captures can be verified by texture synthesis, but texture synthesis also paves the way for a multitude of applications in inpainting, lossy compression, super-resolution, upscaling without distorting the scale of the texture elements (*texels*) and various other image processing techniques. The synthesis of textures is therefore an active research field.

One of the core parametric models before the advent of deep learning was a statistical model proposed by Portilla and Simoncelli [19]. Their approach consists of two phases: an analysis and a synthesis phase. In the analysis phase, image statistics are computed via image decomposition with a steerable pyramid. In the synthesis phase, the algorithm iteratively applies the computed statistical parameters to obtain a texture with the same image statistics. Although the method performs well on a wide range of textures, it does not capture the full scope of irregular textures.

In 2015 Gatys et al. proposed an alternative parametric model based on the feature maps of a convolutional neural network (CNN), laying the groundwork for most neural network applications in texture synthesis [6]. Their method is based on the network architecture VGG-19 and is heavily inspired by Portilla and Simoncelli's algorithm [25]. However, Gatys et al.'s method shows an improved naturalness of synthesised irregular textures over Portilla and Simoncelli's method, as assessed by human observers.

A second subfield of neural texture synthesis that quickly originated after Gatys et al.'s seminal paper is based on the work done by Jetchev et al. on Spatial GANs (SGANs) [12]. SGANs are a modification of Deep Convolutional GANs (DCGANs), which are themselves modifications of GANs [20, 7]. SGANs, like GANs, contain a generator and a discriminator, where the discriminator outputs the probability that

its input came from the data rather than the generator. Unlike GANs, however, a SGAN's generator produces an RGB image. Moreover, its input noise z is not a single vector, but a spatial tensor of size $\mathrm{R}^{l \times m \times d}$. This report will, however, not use this alternative approach.

It is the neural network-based method for texture synthesis proposed by Gatys et al. that provides the foundation for this report [6].

1.3 Contributions

This report provides the following contributions:

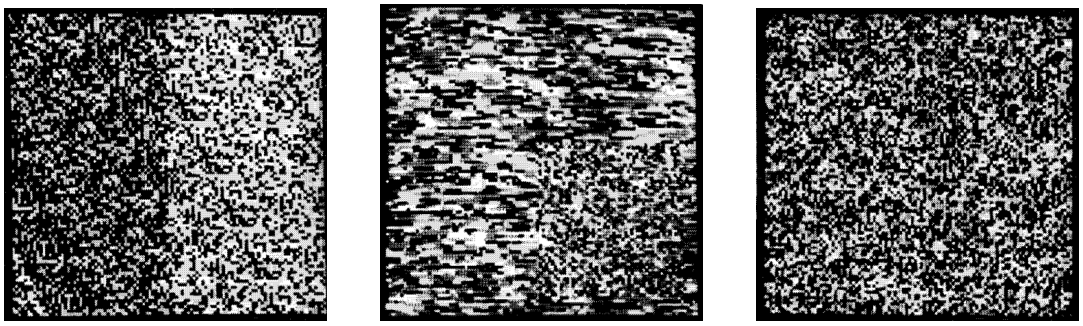
- a **mask-aware** CNN architecture based on VGG-19, containing novel mask-aware convolutional layers and mask-aware pooling layers, enabling texture synthesis using target textures with irregular shapes.
- the combination of mask-aware neural texture synthesis with constrained optimization. This is essentially **inpainting** with neural texture synthesis.
- application of the general idea of neural texture synthesis with Gram matrix representations to a **Vision Transformer** (ViT), first proposed by Dosovitskiy et al. [4].
- **structure-aware texture synthesis** by combining the original texture loss proposed by Gatys et al. with a novel structure loss based on an approximation of the gradient of image intensity [6].

Section 2 expounds on previous work related to the project, after which Section 3 describes the methods and algorithms used in this project. Section 4 presents the results of the proposed methods, which are discussed in Section 5. The project is concluded in Section 6.

2 | Related Work

2.1 Texture perception in humans: Julesz Conjecture

In the 1960s, Béla Julesz investigated the discrimination ability of the human visual system between textures generated by different stochastic processes. These Markov processes were specified by their N-th order joint probability distribution, which is the probability of N selected brightness points having certain values. Figure 2.1 shows three random fields with different first-, second- and third-order probability distributions.



(A) Different first-order probability distributions. Pixels on the left are black with probability $\frac{5}{8}$, while pixels on the right are black with probability $\frac{3}{8}$.

(B) Identical first-order but different second-order probability distributions.

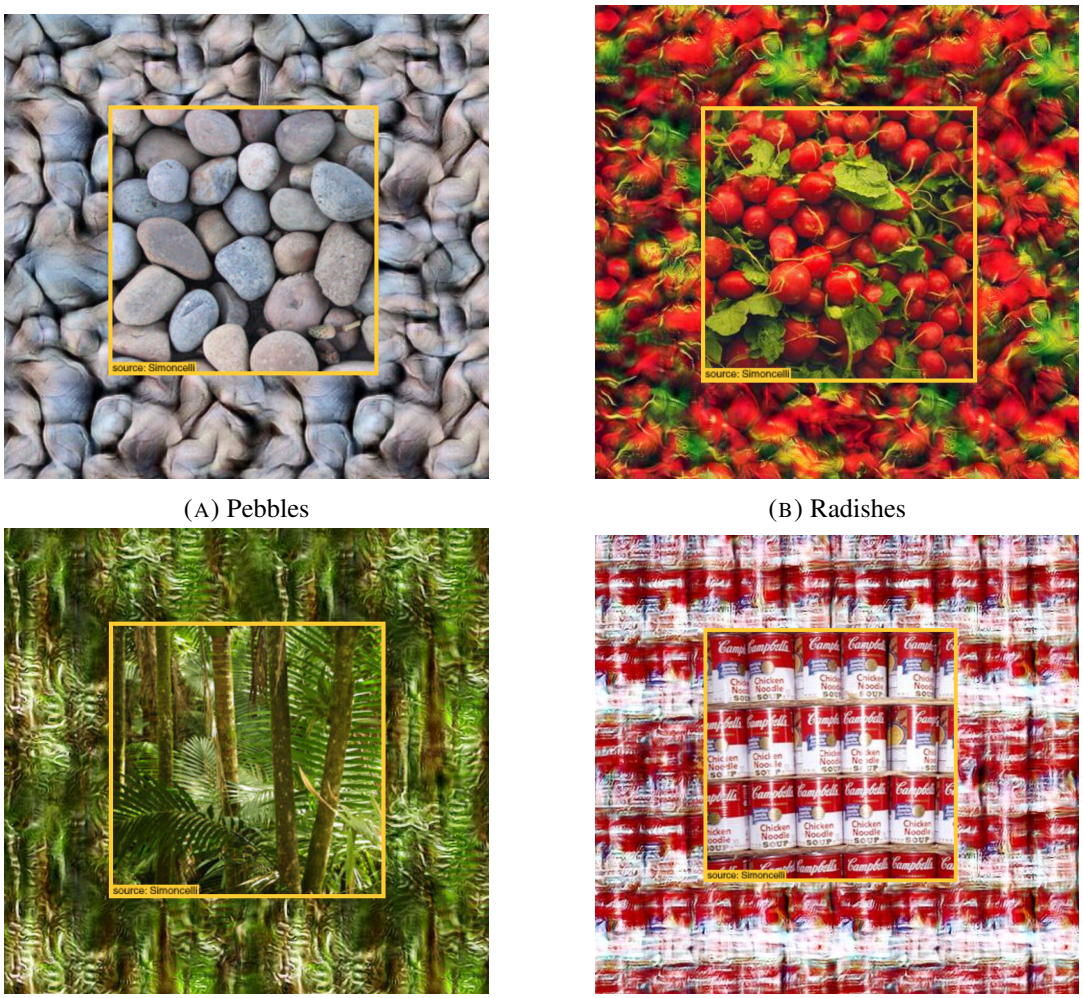
(C) Identical first- and second-order, but different third-order probability distributions.

FIGURE 2.1: Three images containing two random fields with different probability distributions for the first, second and third order. Figure adopted from Julesz [13].

Based on his experimental results, Julesz conjectured that humans can perceive differences only up to second-order statistics. He stated that when distributions are identical up to the second-order and only differing in a higher order, humans would not be able to discriminate between the random fields. However, in 1973 Julesz et al. proved themselves wrong by using non-Markovian processes to generate textures [14]. They found counterexamples where two random fields with different third-order statistics could also be discriminated, suggesting that the human visual system can also differentiate between textures with different higher-order statistics. Nevertheless, the Julesz Conjecture paved the way for the current role of texture statistics in the field of texture synthesis.

2.2 Statistical texture synthesis

Before the use of neural networks for texture synthesis, textures were synthesised and extended with statistical methods. Arguably the best method for natural texture synthesis before CNNs was proposed by Portilla and Simoncelli in 2000 [19].



(A) Pebbles (B) Radishes

(C) Jungle (D) Cans

FIGURE 2.2: Four natural textures extended with the statistical method proposed by Portilla and Simoncelli [19]. Figures adopted from Portilla and Simoncelli [18].

Their method is based on joint statistics of complex wavelet coefficients. The computed statistics function as constraints to synthesise a new instance of the same texture.

The method consists of two phases, a phase where the target statistics are computed and a texture synthesis phase. In the first phase, the statistical measurements are computed by decomposing the original texture with a Steerable Pyramid [24]. A steerable pyramid decomposes an image linearly in both scale and orientation. Figure 2.3 contains a schematic representation of a steerable pyramid. There are two main parameters that can be controlled: the number of levels and the number of subbands

per level. The levels differ on scale, while the subbands have different orientations. An image of an example pyramid decomposition is shown in Figure 2.4.

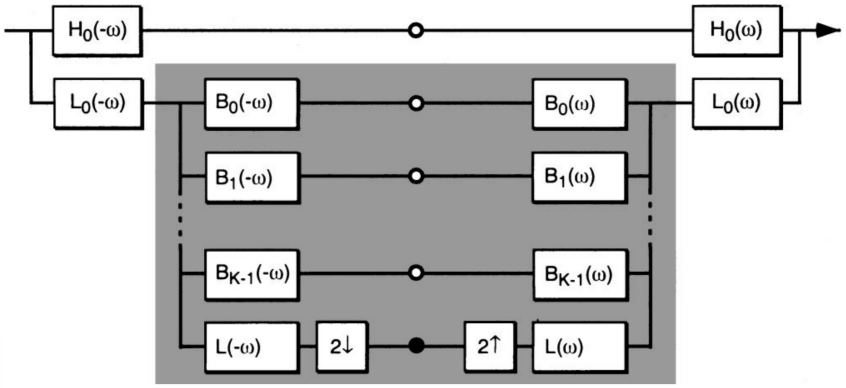


FIGURE 2.3: A schematic representation of a steerable pyramid [24]. The gray area is a recursive subsystem, representing only one level $L_i(-\omega)$. $H_0(-\omega)$ represents the highpass band, $B_i(-\omega)$ is the $(i+1)$-th oriented subband of a given pyramid layer and K is the number of oriented subbands per layer. $L(-\omega)$ is the residual lowpass subband. Figure adopted from Portilla and Simoncelli [19].

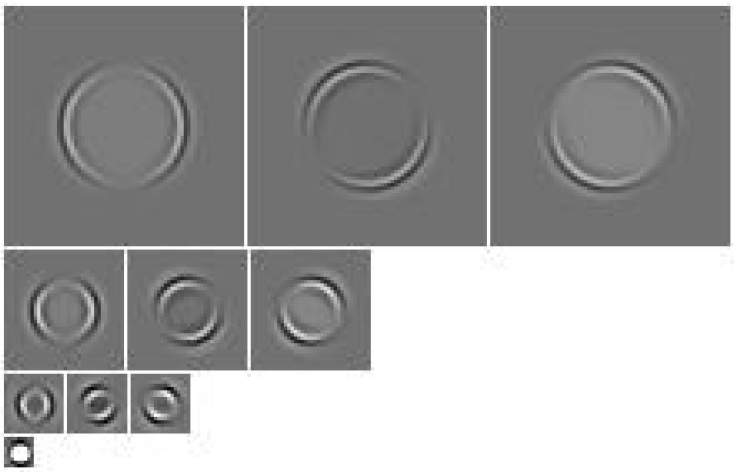


FIGURE 2.4: An example of values in a steerable pyramid with 3 levels and 3 subbands, applied to an image with a black background and a white circle in the middle. Figure adopted from Simoncelli and Freeman [24].

In the synthesis phase, the algorithm starts with Gaussian noise and successively applies statistical parameter constraints. The input image is decomposed with a Steerable Pyramid, after which the statistical measurements are taken on the obtained coefficients. There are four types of statistical constraints:

- **Coefficient correlations.** Coefficient auto-correlations in the low-pass band represent the most important spatial frequencies and regularities.

- **Magnitude correlations.** Auto-correlations of the magnitudes of each subband and cross-correlations of subband magnitudes represent structures, such as edges and corners.
- **Cross-scale phase statistics.** The real and imaginary parts of the complex coefficients are cross-correlated with the next scale.
- **Marginal statistics.** Statistics are constrained directly on pixel values, such as the statistical moments mean, variance, skewness and kurtosis of the high-pass band.

First, coefficient auto-correlations in the reconstructed image with only the low-pass band are enforced, representing the most important spatial frequencies and regularities. Next, the algorithm constrains auto-correlations of the magnitudes of each subband and cross-correlations of subband magnitudes, representing structures, such as edges and corners. Thirdly, cross-scale phase statistics are checked, meaning that the real and imaginary parts of the complex coefficients (the phases) are cross-correlated with the next scale. In the end, marginal statistical properties are imposed, such as skewness and kurtosis of the pixel values and variance of the high-pass band.

These four types of constraints comprise the parameter set. This parameter set is applied recursively, until a suitable image has been reconstructed. The synthesis method is visualised in Figure 2.5.

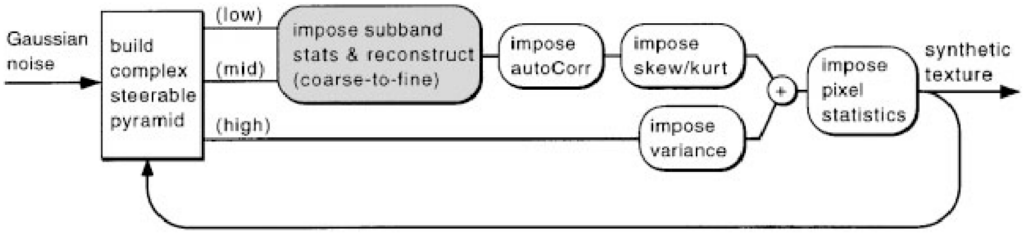


FIGURE 2.5: Diagram of the recursive texture synthesis algorithm proposed by Portilla and Simoncelli [19]. Adopted from Portilla and Simoncelli [19].

2.3 Neural texture synthesis

The extensions in this report are based on the original neural texture synthesis method first proposed by Gatys et al. [6]. The translation equivariant nature of convolutions are especially suited for capturing the repeating patterns of textures. The basic idea behind their method is employing a pre-trained convolutional neural network (CNN) to parameterise the texture. More specifically, the Gram matrices of the network's feature maps contain the representation of a texture.

A schematic overview of the approach in this paper is shown in Figure 2.7. The authors used the first 16 convolutional layers and 5 pooling layers of a VGG-19 network pre-trained on ImageNet [23, 25].

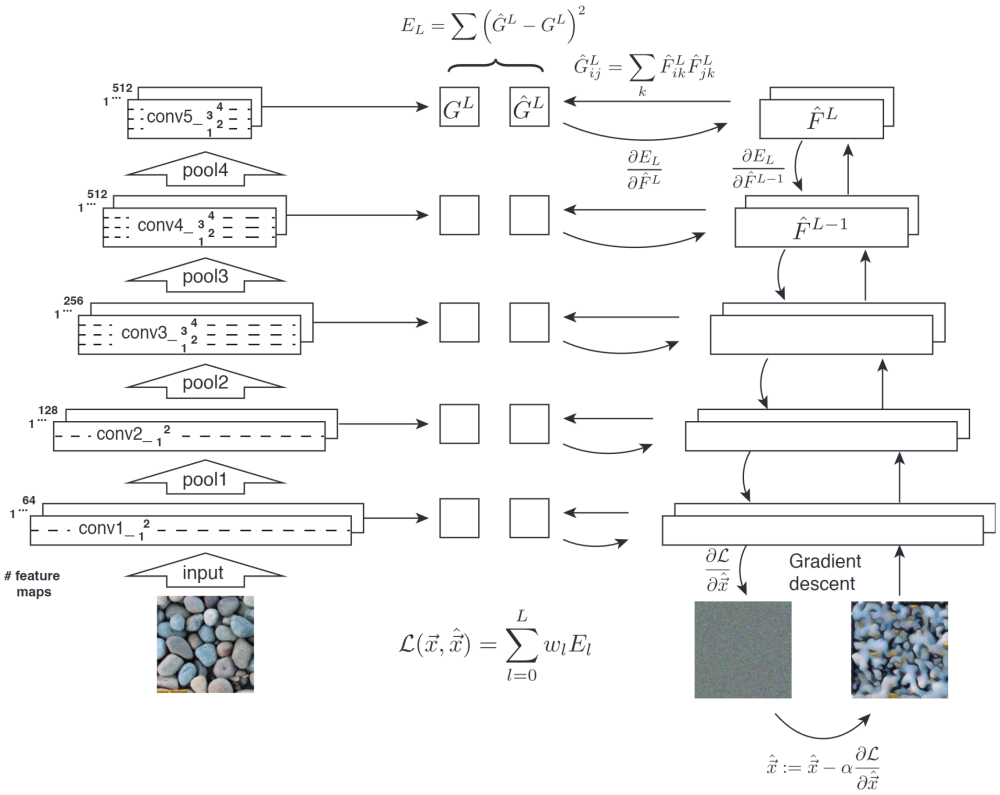


FIGURE 2.6: A schematic overview of the neural synthesis method used in Gatys et al. [6]. The left side visualises the feature maps extracted from VGG-19 with the target texture as input. The right side visualises the extracted feature maps using the synthesised texture as input. Each layer error is computed using the two Gram matrices computed on the feature maps from the left and right sides, which are combined into one weighted loss $L(\vec{x}, \hat{\vec{x}})$. Adopted from Gatys et al. [6].

Elements of the neural network are defined mathematically as follows. A layer l has N_l distinct feature maps of size M_l. The layer's feature maps can therefore be stored in one single matrix F^l of size $N_l \times M_l$.

2.3.1 Gram matrix representations

The Gram matrix $G^l \in \mathbb{R}^{N_l \times N_l}$ contains the dot products between all feature maps in a layer l. For two feature maps F^l_i and F^l_j, the value at row i and column j of G^l is therefore

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}. \tag{2.1}$$

Note that this is the same as the dot product $G^l = F^l (F^l)^T$.

Though we could also directly focus on generating a natural texture with the same feature maps in the CNN as the target image, Gatys et al. instead chose to compute

the Gram matrix on the feature maps first. An advantage of this is that the description of the natural texture is stationary and does not depend on the location of the specific texture features in the provided target image. Another advantage is that the Gram matrix is not dependent on the size of the image, but only on the number of features N_l in each layer of the network. As a result, we can scale the synthesised image as we like without changing the resolution of the natural texture.

2.3.2 Loss function

In order to be able to synthesise a texture, we need a loss function that we can optimise and which is therefore differentiable. Gatys et al. define the loss between target image $\vec{x}$ and synthesised image $\hat{\vec{x}}$ as

$$L(\vec{x},\hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l, \tag{2.2}$$

where w_l are the weighting factors and the layer loss E_l is

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - \hat{G}_{ij}^l\right)^2. \tag{2.3}$$

Note that M_l is the normalization factor for a Gram matrix that is computed with vectorised feature maps of length M_l and N_l^2 is the normalization factor for the sum of all values in a Gram matrix $G^l \in \mathbb{R}^{N_l \times N_l}$.

The prerequisite that the loss is differentiable is upheld, because each layer's loss is differentiable with respect to the layer's feature map.

$$\frac{\partial E_l}{\partial \hat{F}_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2}\left((\hat{F}^l)^T (G^l - \hat{G}^l)\right)_{ji} & \text{if } \hat{F}_{ij}^l > 0, \\ 0 & \text{if } \hat{F}_{ij}^l < 0. \end{cases} \tag{2.4}$$

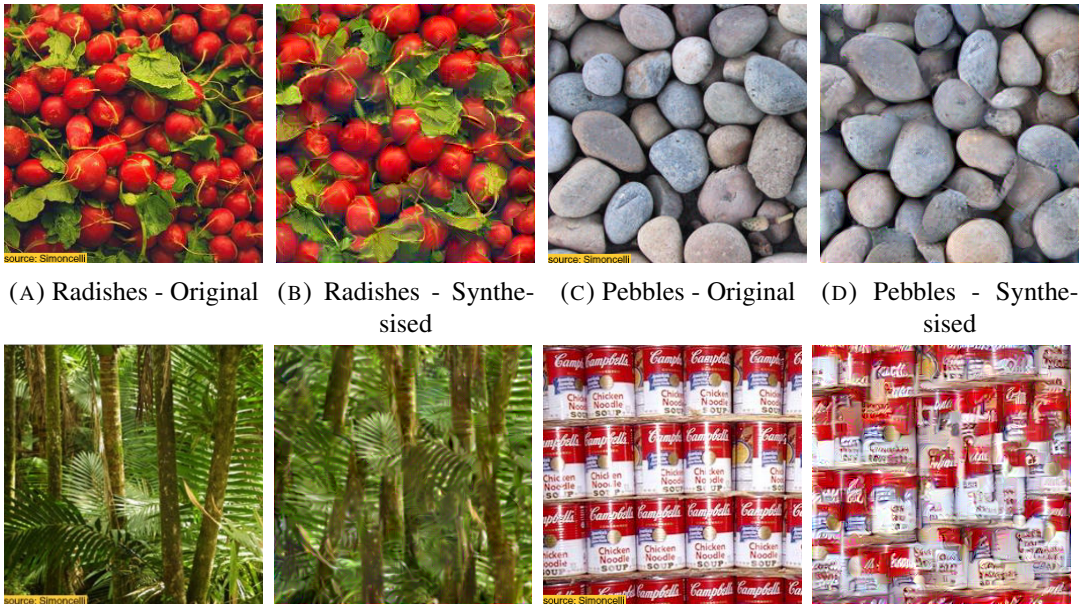
With the gradients of E_l and back-propagation, we can then compute $\frac{\partial L(\vec{x},\hat{\vec{x}})}{\partial \hat{\vec{x}}}$. This is the gradient that enables the optimisation of the synthesised image $\hat{\vec{x}}$ with a numerical optimisation method. The authors used Limited-memory Broyden-Fletcher-Goldfarb-Shanno with bound constraints (L-BFGS-B) for this purpose [2]. Though the analytic gradient is complex and a product of many gradients due to the chain rule, its computation can be simplified by using automatic differentiation, a method implemented in most machine learning libraries.

The pretrained VGG-19 network was trained on colour images of 224 by 224 pixels. When synthesising an image with the same size, this means that one input has $224 \cdot 224 \cdot 3 = 150,528$ parameters. Moreover, pixel intensity is normalized to $[0,1]$ and continuous in order to be differentiable. In other words, the synthesised image is $\hat{\vec{x}} \in \mathbb{R}^{150,528}$. Therefore, the loss landscape has 150,528 dimensions. The original image is the point in the loss landscape of the theoretical global minimum.

2.3.3 Alterations to VGG-19

Gatys et al. made two alterations to the original VGG-19 network. The first is the replacement of the original max pooling layers by average pooling layer. The reason for this is that the authors found that the gradient flow improved with average pooling layers and in addition the results look slightly cleaner [6].

Secondly, the pretrained weights were scaled such that the mean activation of each filter over images is equal to one. This rescaling can be done without changing the output when the scaling factor of the weights in one layer (i.e. by a scaling factor s) is used to divide the weights in a connected layer (i.e. by $1/s$). The mean activations with which to rescale the network can be found by averaging the activations of the network on a given dataset, such as the Describable Textures Dataset (DTD) [3].



(A) Radishes - Original (B) Radishes - Synthesised (C) Pebbles - Original (D) Pebbles - Synthesised

(E) Jungle - Original (F) Jungle - Synthesised (G) Cans - Original (H) Cans - Synthesised

FIGURE 2.7: Four natural textures synthesised with the neural method proposed by Gatys et al. [6]. Sorted by increasing long-range structure, such as the uninterrupted horizontal lines of shelves in Figure 2.7g which span the entire image.

2.3.4 Advantages and limitations

The neural texture synthesis method proposed by Gatys et al. was heavily inspired by the parametric model for texture synthesis by Portilla and Simoncelli. The two main similarities are:

- **Hierarchical architecture.** The VGG-19 architecture has a pyramid architecture for feature extraction, even though the convolutional filters in each layer of the neural network differ from the complex wavelet transformations in the steerable pyramid. Both architectures downsample the image by a factor of $\frac{1}{2}$ a couple of times. It has been said that this hierarchical processing is in line with the human visual system [6].

- **Statistics on coefficients.** The Gram matrix is essentially an approximation of correlations between filter activations (i.e. different subbands within one level). This is highly similar to the magnitude correlations computed in the parametric texture synthesis method [19].

On the other hand, the neural texture synthesis method does not include correlations across scales (i.e. layers of the neural network) or marginal statistics of pixel values. Indeed, when analysing the histogram of pixel values of a texture synthesised with the method by Gatys, one can see that the histogram does not match the original texture. In order to account for this, the authors perform a post-processing step of histogram matching. Other authors have suggested using an alternative representation to Gram matrices by computing the Sliced Wasserstein distance [9]. This makes post-processing with histogram matching unnecessary, but it comes at a computational cost.

Another limitation of the method is that long-range and larger structures in the texture, such as the straight lines in stacked cans or the tree trunks in Figure 2.7, are not handled well. The statistical parametric synthesis method by Portilla and Simoncelli seems better suited to include these structures, as can be seen in Figure 2.2. Additionally, neural texture synthesis is quite computationally expensive, especially compared to statistical methods.

Despite these limitations, the results of the neural synthesis method are still perceived as more natural by the average human observer. A big advantage of the method is that no training of the neural network is needed, due to the use of a pretrained recognition network. The only optimization that occurs is the iterative improvement of a Gaussian noise input with the gradient of the loss function with respect to the input image.

The method works especially well on a smaller scale. With improvements on long-range structure of the synthesised textures, this texture synthesis method has the potential to outperform the statistical parametric method in most aspects of image quality.

2.4 Inpainting

Image inpainting is the practice of reconstructing missing pixels in an image in a by humans undetectable manner [27].

by generating a patch of an image that seamlessly merges into the original image and preferably is indistinguishable from a real image by the human eye.

2.4.1 Partial convolution

In the literature, when inpainting is done with convolutional neural networks, it is often done with partial convolution layers [16]. The main difference with normal convolutions lies in the presence of a *mask*. Next to the activations, each partial convolutional layer also edits and passes on a mask. For weights $\mathbf{W}$, an input $\mathbf{X}$ and a mask $\mathbf{M}$, the output of a partial convolutional layer $\mathbf{x}'$ is

$$\mathbf{x}' = \begin{cases} \mathbf{W}^T(\mathbf{X} \circ \mathbf{M}) \frac{\text{sum}(\mathbf{1})}{\text{sum}(\mathbf{M})} + b & \text{if sum}(\mathbf{M}) > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{2.5}$$

where $\circ$ denotes element-wise multiplication, i.e. the Hadamard product. In the mask $\mathbf{M}$, a value of 1 indicates that the pixel is included in the input, while 0 means it is masked.

After a convolution with Equation 2.5, the mask is updated with

$$\mathbf{M}'_{ij} = \begin{cases} 1 & \text{if } (\mathbf{M} \circ \mathbf{1})_{ij} > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

In the paper, this partial convolution operation was applied in a U-Net-like 'hourglass' architecture trained on a large dataset of images, unlike the method by Gatys et al. [6].

2.5 Receptive field analysis

The receptive field of a convolutional neural network can be computed with [21]

$$r_l[l] = r_l[l-1] + (k_l[l]-1) \prod_{i=0}^{l-1} g_l[i], \tag{2.7}$$

where $r_l[i]$ is the receptive field of layer i, $k_l[i]$ is the filter size of layer i and $g_l[i]$ is the stride in layer i. The receptive layers of both convolutional and pooling layers adhere to this equation. The receptive field of the image itself, before any operations of the neural network, is $r_l[0] = 1$.

2.6 Transformer

The Transformer model is a recent addition to the field of deep learning [28]. First proposed in 2017, the literature around the new neural network architecture has grown considerably in the few years since its conception. Its main contribution is the addition of an attention mechanism to neural networks, which draws global dependencies between the input and output. Figure 2.8 depicts the Transformer architecture.

The Transformer uses a self-attention mechanism. Like most other sequence-to-sequence models, it uses an encoder-decoder structure. The encoder transforms an input sequence of symbol representations $(x_1, ..., x_n)$ into a sequence of continuous representations $\mathbf{z} = (z_1, ..., z_n)$, i.e. the latent space representation. Based on $\mathbf{z}$, the decoder then generates an output sequence $(y_1, ..., y_m)$, where $m > n$.

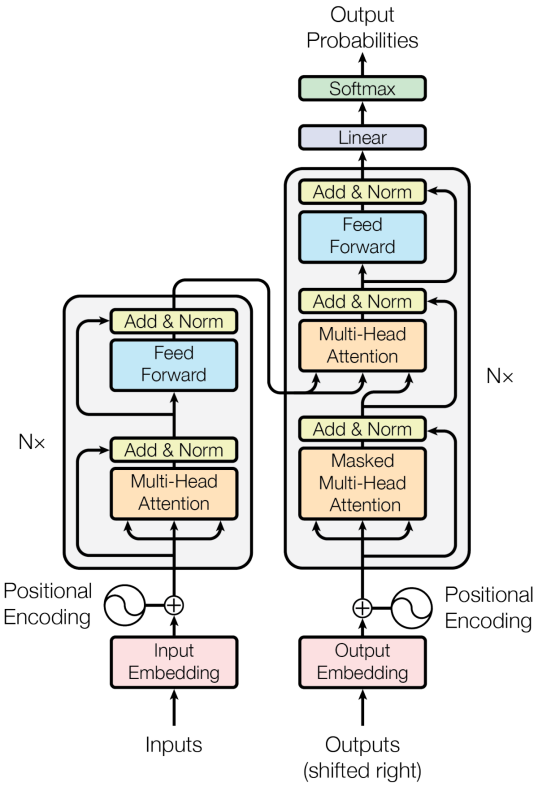


FIGURE 2.8: The Transformer model architecture. Adopted from Vaswani et al. [28].

2.6.1 Encoder and decoder blocks

The encoder and decoder of a transformer are build by stacking N encoder blocks and N decoder blocks on top of each other, respectively.

The encoder block contains two sub-layers: a so-called multihead self-attention layer followed by a position-wise fully connected neural network. In addition, each sub-layer is surrounded by a residual connection like the ones used in ResNets.

The decoder block contains three sub-layers. Next to the multihead self-attention layer and the fully connected neural network in the encoder block, it also contains a third sub-layer that performs multihead attention on the output of the entire encoder.

2.6.2 Scaled dot-product attention

The self-attention operator is at the core of the Transformer architecture and the main reason for its effectiveness. To get an intuitive idea of what self-attention is, we can take a look at movie recommendations.

Let us take a situation where we want to check if a movie $\mathbf{m}$ should be recommended to a user $\mathbf{u}$. Both the movie and the user are represented by vectors of the same length, representing facts like 'is this movie a comedy?' and 'does this user like comedy?'. A way of computing the similarity between the movie and the user's preference is by taking the dot product.

The attention mechanism used is called *scaled dot-product attention*. Its input consists of three elements: queries q, keys k and values v. The user in the example above represents a query and the movie is similar to a key. The naming is based on the concept of key-value pairs in dictionaries. A key is connected to a certain look-up value in a dictionary. However, in our case queries don't exactly match the keys of a dictionary, so instead we can compute a similarity score between the query and all possible keys, to generate an aggregated look-up value. In essence, attention is a linear transformation of the values, where the coefficients are a function of the queries and keys.

To avoid exploding gradients, the dot product is scaled by $\sqrt{d_k}$, where d_k is the length of the query q and key k, to ensure the variance is equal to the input variance.

When multiple queries, keys and values are packed together in matrices Q, K and V to speed up the calculations, the attention mechanism is

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.8}$$

The softmax function maps the values to a probability interval [0, 1] that sum to 1.

2.6.3 Multi-head attention

The scaled dot-product attention is extended into a *multi-head attention* mechanism. Each of the h heads $head_i$ employ the single attention mechanism in equation 2.8, which are concatenated as

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_h)W^O. \tag{2.9}$$

where $head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, of which the projections are parameter matrices W_i^Q, W_i^K and W_i^V, such that for the same input each head pays attention to a different part of the input.

This multi-head attention mechanism is used as a *self-attention* mechanism in one sub-layer of both the encoder and decoder blocks. Self-attention describes the fact that both the query, key and value are the same. In the transformer architecture, this is the output vector of the previous encoder or decoder block.

2.6.4 Position

It is important to note that the transformer model does not look at the input sequence as a sequence per se: the positions of each element in the sequence are not taken into account. This can amongst others be seen in the second and third sub-layers of the encoder and decoder blocks, respectively. These are fully connected neural networks that are applied to each position of the input sequence in the same way, of the form

$$f(x) = \max(0, xW_1 + b_1)W_2 + b_2, \tag{2.10}$$

where x is the input activation map, W_1 and W_2 are weights and b_1 and b_2 are biases.

To ensure that the transformer does take position into account, a positional encoding is added to the input embeddings of both the encoder and decoder stacks. In the original transformer paper, these positional encodings are sines and cosines

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}),$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}),$$

where pos is the position in the sequence, i is the dimension in the embedded vector, d_{model} represents the size of the output vectors of all layers and $1 \leq i \leq d_{model}$. In the original paper, $d_{model} = 512$.

2.6.5 Vision Transformer

Though the Transformer architecture was initially developed with natural language processing tasks in mind, a new architecture based on the Transformer called the Vision Transformer (ViT) was adapted to handle 2D visual inputs [4]. An overview of its architecture is shown in Figure 2.9.

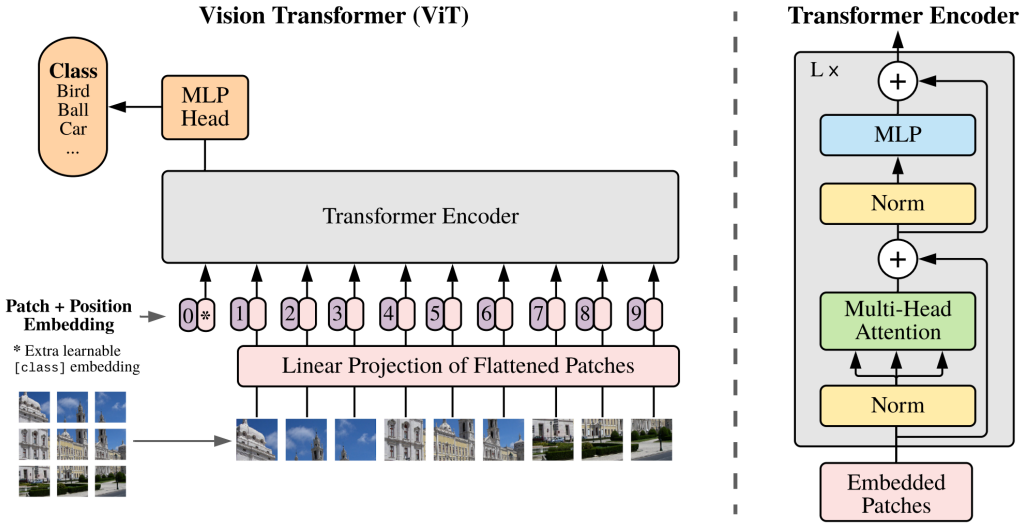


FIGURE 2.9: An overview of the Vision Transformer architecture. An image is split into fixed-size patches. Each patch is linearly embedded and position encoding is added. The result is fed to L consecutive standard Transformer Encoder blocks. In order to perform classification, an extra MLP head is added at the end. Adopted from Dosovitskiy et al. [4].

Whereas in a basic Transformer input tokens consist of words, a ViT requires patches of an image as input, to which it adds a position embedding. Moreover, the ViT does not use decoders, but only L consecutive encoders.

Inductive bias An inductive bias in the context of machine learning refers to an assumption of a learning algorithm in order to perform induction. In this case, induction refers to the ability to generalize what the model learned from training data. That means that certain mechanistic choices influence a learning algorithm's generalization ability. Without these mechanistic choices, multiple types of generalizations can be made. A learning algorithm therefore always has to have an inductive bias [11, 17].

The inductive biases of Vision Transformer is different from the inductive biases of CNNs [27]. A CNN has three major inductive biases: locality, a 2D neighbourhood structure and translation equivariance due its convolution operations. ViT does not have these biases and therefore do not generalize well when trained on insufficient amounts of data [4]. In general, ViT has a much less image-specific inductive bias than CNNs. In ViT only the multi-layer perceptron layers exhibit the inductive biases of locality and translation equivariance, but two-dimensional neighbourhood structure is used sparingly. On the other hand, the self-attention layers in ViT have the additional inductive bias of globality [4].

3 | Method

The project was implemented in Python and PyTorch on a computer with Windows 11 Pro. A preview version of PyTorch with GPU support using CUDA 12.1 was installed. The code is stored in a private GitLab repository and can be made available upon request. The original method by Gatys et al. was reproduced and built upon as a starting point.

3.1 Mask-aware neural texture synthesis

Inpainting is often performed with neural networks *trained* specifically for the task of inpainting. An example of a network architecture used in such cases is the U-Net neural network architecture, first developed for biomedical image segmentation [22]. This U-Net architecture is also used in diffusion probabilistic models in the denoising steps and as such this type of network architecture is the backbone of many modern image generation methods [10].

These models are huge and take massive amounts of time to train on enormous datasets. Additionally, the models generally require textual input to generate images, which is not always the preferred method.

A big advantage of the neural texture synthesis by Gatys et al. is therefore that it does not need training specific for texture synthesis, but instead it can be performed directly with a neural network pretrained as an image recognition network.

It has been shown that constrained optimization in the context of neural texture synthesis is possible [26]. When we want to convert the neural texture synthesis method proposed by Gatys et al. to also apply to inpainting, we could use a similar constrained optimization method, where we hold static the pixels of the original image we want to retain and optimise the remaining pixels, such that the entire synthesised image has a similar Gram matrix representation as the original image.

The main difficulty lies in the difference between the target image and the synthesised image. Only specific pixels in the target image are relevant for texture synthesis. If the entire image were used as the target image, an irregularity in the texture (such as a leaf on a texture of pebbles) is simply copied in the synthesised image. If one wanted to remove this irregularity, this method would not work. Instead, to remove any part of the image and fill it with the specified texture, only selected pixels that contain the wanted texture need to be included in the texture representation.

A solution to this problem could be a mask-aware network, that can focus on the relevant parts of an image containing the texture and ignore the parts that do not.

3.1.1 Mask-aware VGG-19 for texture synthesis

The VGG-19 network was adapted for texture synthesis by Gatys et al. [6] by only including layers meant for feature extraction in the computation of the loss function. These layers consist of 2D convolutional layers, ReLU activation layers and pooling layers. Though the original VGG-19 architecture used max pooling layers, Gatys et al. replaced these layers by average pooling to improve the gradient flow. Therefore, in transforming their architecture into a mask-aware architecture, the pooling layers will be mask-aware average pooling layers.

There are three types of layers that we need to convert to mask-aware versions: 2D convolutional layers, ReLU activation layers and average pooling layers. The conversion of the ReLU activation layers are straight-forward: because the activation function only applies to a single pixel, the input mask can be returned directly.

Mask-aware convolutions. For the conversion of 2D convolutional layers into mask-aware 2D convolutional layers, the partial convolutional layers proposed by Liu et al. [16] are taken as a starting point, but directly using these layers causes *exploding gradients* in areas with masked pixels. The reasons for this is that the scaling factor $\text{sum}(\mathbf{1})/\text{sum}(\mathbf{M})$ mentioned in Equation 2.5 causes the influence of the unmasked pixels to increase by this scaling factor. This means that the gradients of these pixels are also increased in the same ratio, which is a problem when the scaling factor has a value larger than 1. Because this increased influence is not only perpetuated in the network, but also increased in each convolutional layer by another scaling factor above 1, the effect is a compound effect.

As a result, this scaling factor should be left out to make sure that the influence of each pixel on the final loss is equal. The mask-aware convolution operation is therefore

$$\mathbf{X}' = \begin{cases} \mathbf{W}^T(\mathbf{X} \circ \mathbf{M}) + b & \text{if } \text{sum}(\mathbf{M}) > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

Mask-aware average pooling. Regarding a mask-aware average pooling operation, two ways to perpetuate the masks are compared. When masked and unmasked pixels are clustered together, we can either make the resulting pixel masked or unmasked. We took the following consideration: to preserve the texture that the pooling operation needs to compress without including seemingly black boundary pixels, the resulting pixel should be *masked*. On the other hand,

In Section 2, it is mentioned that masked pixels are represented by 0 and unmasked pixels by 1. With the above consideration in mind, the operation to pool the masked pixels is then effectively min pooling of the input masks. Vice versa, when the combination of masked and unmasked pixels should be *unmasked*, the operation is max pooling of the input masks.

Because min pooling is not implemented in PyTorch, this operation was conducted via max pooling, where for an input mask $\mathbf{M}$

$$\text{MinPool}(\mathbf{M}) = -\text{MaxPool}(-\mathbf{M}). \tag{3.2}$$

The output feature map $\mathbf{X}'$ of the mask-aware average pooling layer should contain the average value of all non-masked pixels in each filter operation. To prevent exploding gradients, by the same reasoning as for mask-aware convolutions we do not make the average pooling dependent on the number of masked pixels. For an input feature map $\mathbf{X}$, the mask-aware average pooling operation is

$$\mathbf{X}' = (\mathbf{X} \circ \mathbf{M}) \frac{1}{\text{sum}(\mathbf{1})} \tag{3.3}$$

Influence on the receptive field. The value of a pixel in the output of the 5th pooling layer in VGG-19 is based on $1 + 2 \cdot (16 + 5) = 43$ pixels in width and height, so $43 \cdot 43 = 1,849$ pixels in total, each containing 3 colour channels. This computation, however, is based on the assumption that the input is a square matrix. When masks are included, the receptive field is less straightforward.

This mainly becomes a problem when the masked pixels are spread evenly over the image and this results in many masked pixels in the higher-level layers with large receptive fields. If there is even one masked pixel among the $1,849$ pixels in the receptive field of the 5th pooling layer, this pixel is also masked. In these cases, the choice can be made to instead perform a MaxPool operation on the feature maps passing through the mask-aware average pooling layers, though it comes with a potential reduction in image quality.

3.1.2 Conversion of a pretrained VGG into a mask-aware VGG

Because ReLU layers and pooling layers do not have weights or biases, these layers do not have to be trained and can simply be replaced by their mask-aware counterparts. The convolutional layers can be converted to mask-aware versions by copying the pretrained weights and biases of the convolutional layers in the standard VGG-19 network pretrained on ImageNet into the mask-aware network [23].

Of course, it would be more ideal to directly train the mask-aware VGG-19 network as a recognition network on a large dataset like ImageNet [23]. However, because it takes multiple weeks on a normal personal computer to train a dataset of more than 14 million images, this alternative route was chosen instead. Training the mask-aware network on the smaller Describable Textures Dataset (DTD) was attempted, but the relatively short training time yielded insufficient network weights and biases for any texture to emerge from the input Gaussian noise during synthesis.

3.1.3 Gram matrix representations in mask-aware VGG-19

Each entry in a Gram matrix is the *mean* product between two feature maps. In other words, the Gram matrix is simply a correlation for each individual position in

a feature map between different filter values. This means the shape of these feature maps does not influence the final result and the Gram matrix does not capture the relation between different positions within the same feature map.

Because of this lack of relationship, certain positions in the feature maps can be left out, which should result in the same static texture representation as before given that the input texture is the same:

$$G_{ij}^{l} = \sum_{k} \mathbf{M}_k F_{ik}^{l} F_{jk}^{l}, \tag{3.4}$$

where $\mathbf{M}_k$ is the value at index k in the vectorised mask $\mathbf{M}$.

The layer loss is based on Equation 2.3, which only differs in the normalization parameter M_l. This term should be replaced with the value sum($\mathbf{M}$), because the length of the feature map has now reduced to sum($\mathbf{M}$).

3.2 Constrained neural texture synthesis

In the paper by Surace et al. [26], the neural texture synthesis method proposed by Gatys et al. has been used in conjunction with a randomly sampled set of constrained pixels in order to additionally preserve the higher-level structures in a natural texture. Their GAN-based method shows that in this way the structure problem can be solved. When we extend this idea of constraining random pixels to manually selected pixels, the problem can amongst others become an image inpainting problem. Especially inpainting of patches where the surrounding natural texture should be extended could benefit from this approach of neural texture inpainting.

When combining this constrained optimization with the mask-aware network described above, texture synthesis accepts a flexible image input shape instead of only the standard rectangular shape, allowing the extraction of irregularly shaped patches of texture from an image and upscaling it to both the wanted size and shape. Moreover, it can be extended into neural texture inpainting, of both rectangularly shaped holes but also irregularly shapes ones. When this method of textures synthesis is combined with texture segmentation using pre-existing texture segmentation networks, the texture in one area of an image could be synthesised in another.

Constraints can naturally be added with the optimisation method L-BFGS-B [2]. The original RGB values of the constrained pixels serve as both lower and upper bounds.

3.3 Structure-aware texture synthesis

In Section 2.3.4 and Figure 2.7 it was mentioned and shown that the neural synthesis method by Gatys et al. has difficulties in synthesising textures that have long-ranged structures and that do not only exist on a small scale. This is in line with the inductive bias of locality present in CNNs, which were mentioned in Section 2.6.5.

We propose two alternative methods to neural texture synthesis that include long-ranged structures. Both are *structure-aware* texture synthesis methods. The main goal is to preserve the structure of the original image while synthesising the texture onto a structural backbone. However, the synthesised image should not be exactly the same as the original. Instead, the texture should be the same to the human eye, but the images should deviate pixel-wise. This is to show that the algorithm did not only reproduce the old image, but was able to synthesise a realistic new image using structure and a static texture representation.

3.3.1 Neural texture synthesis with Vision Transformer

The authors of the Transformer architecture mention that the model is especially good at recognising global dependencies, i.e. longer-ranged structures, due to its self-attention layers [28]. In addition, Portilla and Simoncelli state that in their architecture, the magnitude correlations in each subband represented *structures*, such as edges and corners.

Gatys et al. showed that only including in-place filter correlations was already an accurate texture representation on feature maps extracted from a VGG-19 network. It would be interesting to see if the same holds for the Vision Transformer, and if the Gram matrix representation on ViT's activations indeed captures the long-range structures that are missing from Gatys's approach.

The number of patches of the ViT used for feature extraction were set to $14 \cdot 14 = 196$. The feature maps of all layers were included in the loss function.

3.3.2 Neural texture synthesis with a gradient loss

Like in neural texture synthesis with a loss purely based on Gram matrix representations of feature maps extracted from VGG-19, the global minimum of the loss landscape (or at least one of the global minima) in texture synthesis with a gradient-based loss is the target image. When adding a structure loss that is aimed to build the original structure but with a slightly different texture, it is therefore important that the structural loss *only* contains information about the structure and no information about the texture. The main difference between long-range structure and texture is the *scale* at which these structures occur.

The gradient of an image captures differences in image intensity values, but these image intensity gradients could also be a textural characteristic. However, if the images were blurred and downsampled, only the higher-level structures remain. This forms the basis for a gradient loss function.

For a target image T and a synthesised image S, the gradient loss is defined as

$$L_{grad} = \frac{1}{M^2} \sqrt{(G_x(T) - G_x(S))^2 + (G_y(T) - G_y(S))^2}, \tag{3.5}$$

where M is the width and height of the result of operations G_x and G_y. These operations consecutively:

1. Convert the image to a grayscale image
2. Blur the image with a Gaussian blur filter of size 7×7
3. Downsample the image 2 times by a factor $1/2$

Note that this loss can only be applied when the sizes of T and S are equivalent.

This gradient loss can be combined with Gatys et al.'s Gram matrix loss as:

$$L = L_{Gatys} + \alpha \cdot L_{grad}, \tag{3.6}$$

where α is the weight of the gradient loss.

4 | Results

4.1 Mask-aware texture synthesis

Four cases were researched with the mask-aware VGG-19 network in conjunction with constrained optimisation. When we mention ‘unmasked’, we mean all values in the mask are set to 1. This is effectively normal texture synthesis with a mask-aware neural network.

The four cases are:

1. **Unmasked and unconstrained.** An unmasked target image with an unconstrained synthesised image.
2. **Unmasked and constrained.** An unmasked target image with a constrained synthesised image.
3. **Masked and unconstrained.** A masked target image with an unconstrained synthesised image.
4. **Masked and constrained.** A masked target image with a constrained synthesised image. This is effectively inpainting.

Figures showing the masked target images have black pixels where the mask has zero-valued pixels. The algorithm was tested on ten different textures. This section shows the results on the texture of radishes. For the results of the other textures we refer to the GitHub repository, due to the high amount of synthesised images.

4.1.1 Rectangular mask

All masked cases used an identical mask of a rectangle in the upper-left corner. In these cases, the masked area covered 7.6% of the image. Vice versa, the constrained pixels covered 92.4% of the image in the constrained cases.

TABLE 4.1: Mean squared error between the synthesised and original images over all four cases for a rectangular mask in the upper-left corner of the image.

Image	Case 1		Case 2		Case 3		Case 4	
	Min	Max	Min	Max	Min	Max	Min	Max
radishes	6047	5717	1158	61	6200	5836	1153	828
cans	8987	8915	1209	252	8798	8832	1196	1331
pebbles	5225	5019	930	144	5211	4976	1029	752
jungle	4031	3507	1220	66	4096	3643	1251	569
non-texture	6759	6005	1195	349	6246	5763	1173	1004
bubbly	4037	4111	1094	106	4275	4667	1025	2145
cobweb	7041	6821	1256	420	6903	6876	1226	754
grid	17636	13710	2415	650	18058	13667	2430	629
wrinkled	2438	2380	845	132	2517	2522	817	797
stochastic	2084	1993	776	109	2088	2364	774	1039

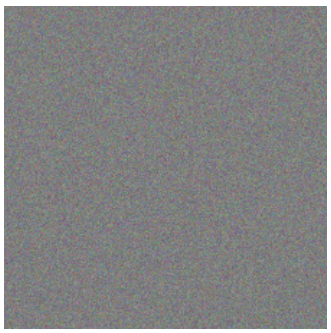
TABLE 4.2: Mean absolute error between the synthesised and original images over all four cases for a rectangular mask in the upper-left corner of the image.

Image	Case 1		Case 2		Case 3		Case 4	
	Min	Max	Min	Max	Min	Max	Min	Max
radishes	62.2	59.7	7.5	1.6	62.8	60.6	7.6	6.2
cans	75.6	75.9	7.8	3.2	74.3	75.7	7.8	8.6
pebbles	57.4	55.7	7.4	2.5	57.1	55.9	7.8	6.5
jungle	52.3	48.4	8.1	1.7	52.4	49.0	8.2	5.5
non-texture	65.0	62.1	8.1	3.9	63.6	61.1	8.0	7.3
bubbly	46.8	46.7	6.9	2.0	47.8	49.8	6.6	11.7
cobweb	67.4	66.7	8.5	4.3	67.3	66.6	8.3	6.2
grid	113.6	98.7	11.3	5.1	115.1	98.4	11.3	5.3
wrinkled	39.6	39.4	7.0	2.4	40.1	40.0	6.8	6.6
stochastic	35.1	34.4	6.7	2.2	35.1	37.4	6.6	7.9

Case 1: unmasked and unconstrained. Figure 4.3 contains the results for an unmasked target image with unconstrained optimisation, given that the mask is updated with *max pooling*. Artefacts such as the white end of a radish can be reproduced almost exactly. This observation is supported by Table 4.1 and Table 4.2, which show relatively low errors, although updating the mask with max pooling in all cases shows lower errors than min pooling. Compared to the results of the original method by Gatys et al. visualised in Figure 2.7, these images are lighter and show a shift to the right in their histograms compared to the original image's histogram.

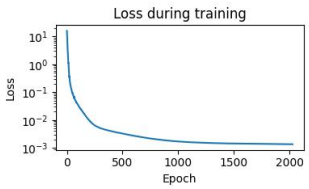
Figure 4.4 contains the results for an unmasked target image with unconstrained optimisation, given that the mask is updated with *min pooling*.

(A) Target

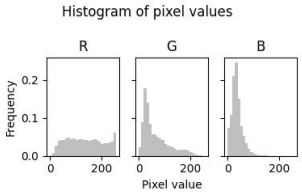


(B) Initial image

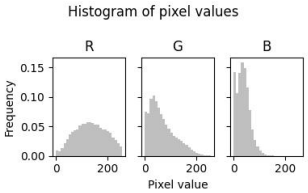
(C) Synthesised



(D) Losses



(E) Original colour distribution



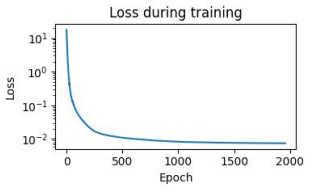
(F) Synthesised colour distribution

FIGURE 4.1: Case 1 applied to radishes. Mask updated with max pooling.

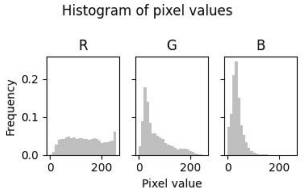
(A) Target

(B) Initial image

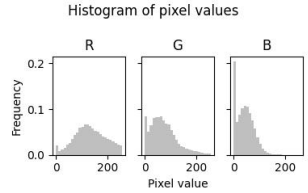
(C) Synthesised



(D) Losses



(E) Original colour distribution



(F) Synthesised colour distribution

FIGURE 4.2: Case 1 applied to radishes. Mask updated with min pooling.

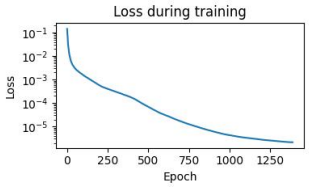
Case 2: unmasked and constrained. Figure 4.3 contains the results for an unmasked target image with constrained optimisation, given that the mask is updated with *max pooling*. As Table 4.1 and Table 4.2 also indicate, the difference between the synthesised texture and the original one is minimal.

Figure 4.4 contains the results for an unmasked target image with constrained optimisation, given that the mask is updated with *min pooling*. With min pooling, the synthesised pixels are lighter than the constrained ones.

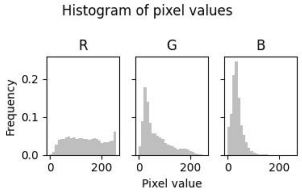
(A) Target

(B) Initial image

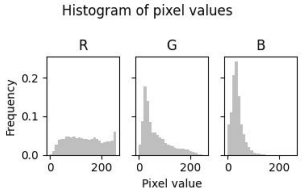
(C) Synthesised



(D) Losses



(E) Original colour distribution



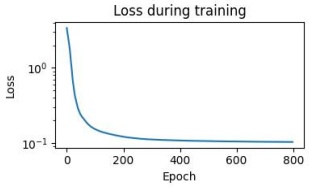
(F) Synthesised colour distribution

FIGURE 4.3: Case 2 applied to radishes. Mask updated with max pooling.

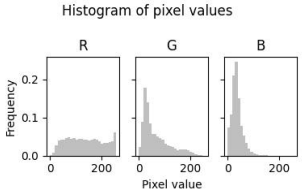
(A) Target

(B) Initial image

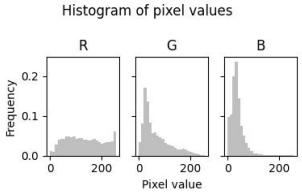
(C) Synthesised



(D) Losses



(E) Original colour distribution



(F) Synthesised colour distribution

FIGURE 4.4: Case 2 applied to radishes. Mask updated with min pooling.

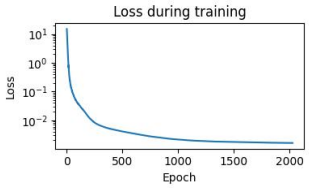
Case 3: masked and unconstrained. Figure 4.5 contains the results for a masked target image with unconstrained optimisation, given that the mask is updated with *max pooling*.

Figure 4.6 contains the results for a masked target image with unconstrained optimisation, given that the mask is updated with *min pooling*.

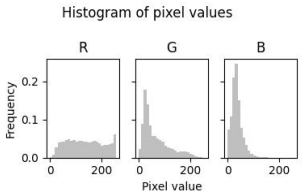
(A) Target

(B) Initial image

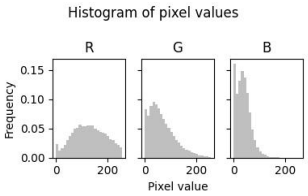
(C) Synthesised



(D) Losses



(E) Original colour distribution



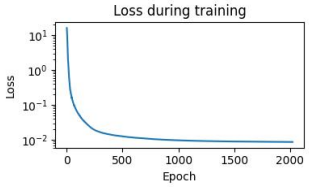
(F) Synthesised colour distribution

FIGURE 4.5: Case 3 applied to radishes. Mask updated with max pooling.

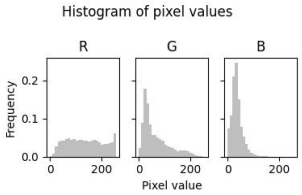
(A) Target

(B) Initial image

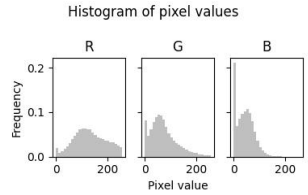
(C) Synthesised



(D) Losses



(E) Original colour distribution



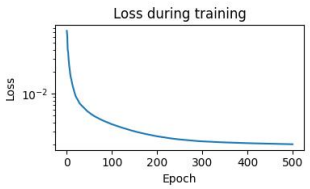
(F) Synthesised colour distribution

FIGURE 4.6: Case 3 applied to radishes. Mask updated with min pooling.

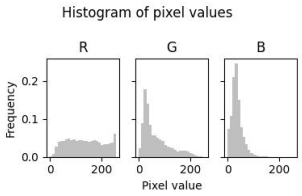
Case 4: masked and constrained. Figure 4.7 contains the results of a masked target image with constrained optimisation, given that the mask is updated with *max pooling*. The results do not contain a reconstruction of the original texture, though a very slight low-level structure appears. However, the colours are darker and not based on the original texture.

Figure 4.8 contains the results of a masked target image with constrained optimisation, given that the mask is updated with *min pooling*. The texture of the target image emerges in the area that is being synthesised, but the colour distribution does not accurately represent the original image's. Instead, colours look lighter and are clearly distinguishable from the original texture's colours.

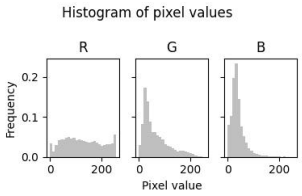
(A) Target (B) Initial image (C) Synthesised



(D) Losses



(E) Original colour distribution

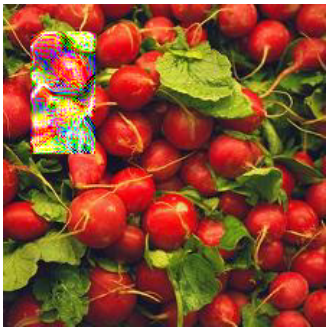


(F) Synthesised colour distribution

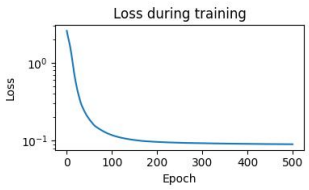
FIGURE 4.7: Case 4 applied to radishes. Mask updated with max pooling.

(A) Target

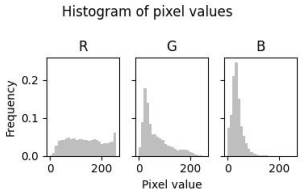
(B) Initial image



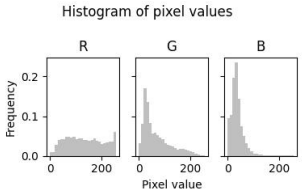
(C) Synthesised



(D) Losses



(E) Original colour distribution



(F) Synthesised colour distribution

FIGURE 4.8: Case 4 applied to radishes. Mask updated with min pooling.

4.1.2 Random mask

We generated a random mask in which each pixel had a probability of 0.5 of having a value of one. The constraints were the inverse of the mask, i.e. a one in the mask was a zero in the binary image with constraints and vice versa.

We only look into cases 2 to 4, because case 1 is identical to the case presented in Section 4.1.1.

TABLE 4.3: Mean squared error between the synthesised and original images over all four cases for a random mask with probability 0.5.

Image	Case 2		Case 3		Case 4	
	Min	Max	Min	Max	Min	Max
radishes	1058	538	6813	6497	1390	4455
cans	1316	770	10774	13231	2532	11376
pebbles	569	207	6782	7896	683	7136
jungle	617	314	3720	4651	1022	3530
non-texture	1283	614	9362	8395	940	6819
bubbly	898	627	8132	15629	1481	15326
cobweb	1076	314	8625	8150	1623	6250
grid	2548	342	9144	13051	1888	2620
wrinkled	535	137	4172	6421	729	6655
stochastic	471	279	3641	8980	976	8890

TABLE 4.4: Mean absolute error between the synthesised and original images over all four cases for a random mask with probability 0.5.

Image	Case 2		Case 3		Case 4	
	Min	Max	Min	Max	Min	Max
radishes	17.7	12.6	62.1	66.7	19.5	37.5
cans	19.2	14.5	82.5	93.9	26.1	65.7
pebbles	12.8	7.7	65.7	73.2	12.4	52.0
jungle	13.2	9.5	45.9	56.2	15.7	34.6
non-texture	19.0	12.8	77.6	74.0	14.6	48.3
bubbly	15.2	12.7	64.2	98.4	18.2	76.7
cobweb	17.8	9.4	75.6	72.8	21.1	46.6
grid	22.5	6.6	86.7	102.9	15.1	19.4
wrinkled	12.6	6.3	51.6	64.8	13.2	50.4
stochastic	11.7	9.0	46.6	78.1	16.0	61.0

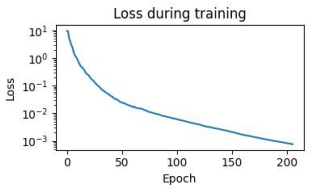
Case 2: unmasked and constrained. Figure 4.9 contains the results for an unmasked target image with constrained optimisation, given that the mask is updated with *max pooling*. Figure 4.10 contains the results for an unmasked target image with constrained optimisation, given that the mask is updated with *min pooling*. It can be seen that the main difference between these two settings is the synthesis near the borders of the image.

For all images that case 2 was applied to in both Table 4.3 and Table 4.4, case 2 in combination with max pooling to update the mask reach the lowest reproduction errors and come closest to the global minimum that is the original texture.

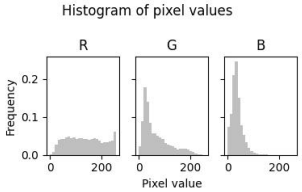
(A) Target

(B) Initial image

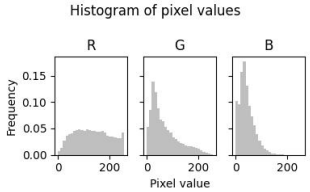
(C) Synthesised



(D) Losses

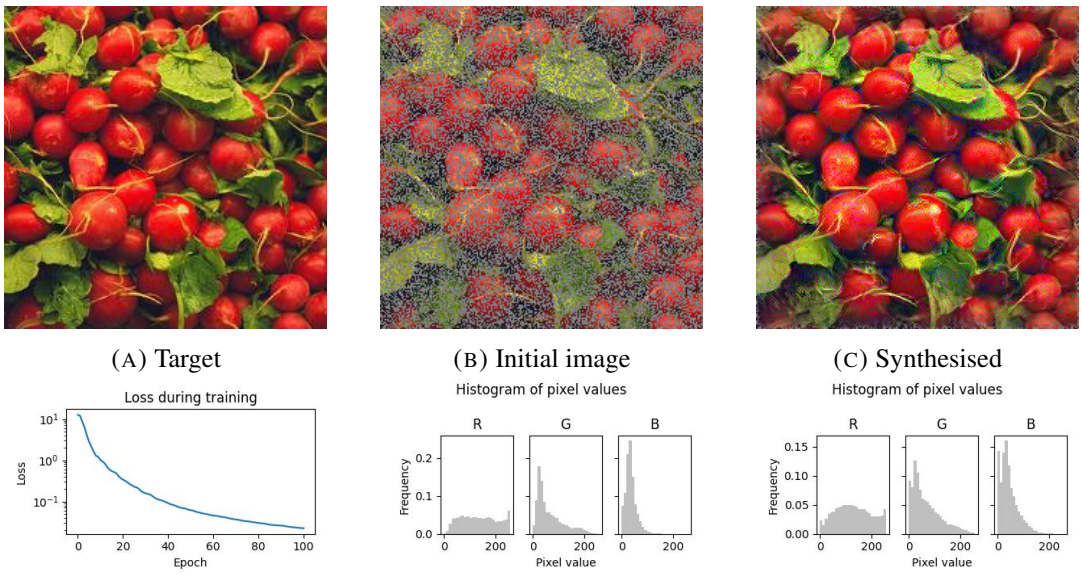


(E) Original colour distribution



(F) Synthesised colour distribution

FIGURE 4.9: Case 2 applied to radishes. The mask is randomly generated with a probability of 0.5. Mask updated with max pooling.



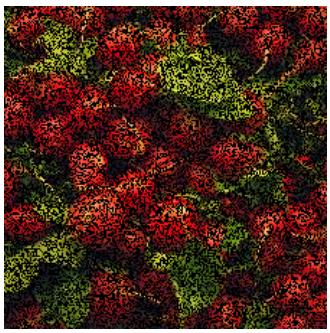
(A) Target (B) Initial image (C) Synthesised

(D) Losses (E) Original colour distribution (F) Synthesised colour distribution

FIGURE 4.10: Case 2 applied to radishes. The mask is randomly generated with a probability of 0.5. Mask updated with min pooling.

Case 3: masked and unconstrained. Figure 4.11 contains the results for an unmasked target image with constrained optimisation, given that the mask is updated with *max pooling*. The masked target texture does not seem sufficient to capture the original texture. The colour distribution is shifted to the right with respect to the original colour distribution and the texture of the original image does not emerge based on the masked target during the unconstrained optimisation.

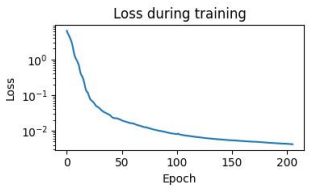
Figure 4.12 contains the results for an unmasked target image with constrained optimisation, given that the mask is updated with *min pooling*. This case shows the least resemblance to the original texture. The colour distribution has shifted to the left with respect to the original colour distribution and the texture of the original image does not emerge based on the masked target during the unconstrained optimisation.



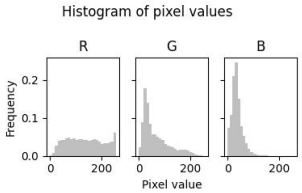
(A) Target

(B) Initial image

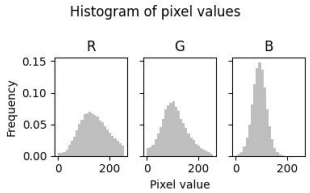
(C) Synthesised



(D) Losses

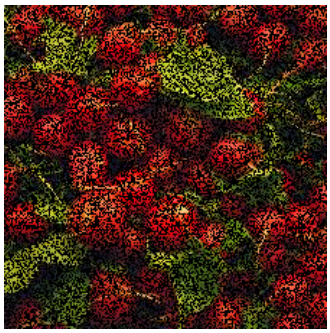


(E) Original colour distribution



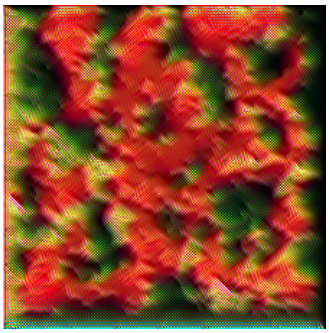
(F) Synthesised colour distribution

FIGURE 4.11: Case 3 applied to radishes. The mask is randomly generated with a probability of 0.5. Mask updated with max pooling.

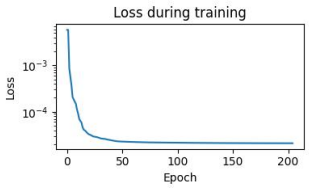


(A) Target

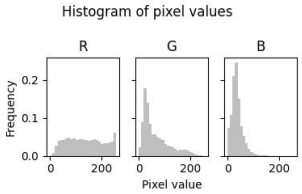
(B) Initial image



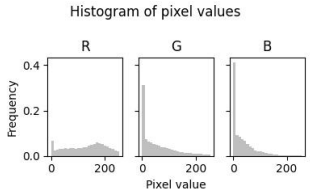
(C) Synthesised



(D) Losses



(E) Original colour distribution

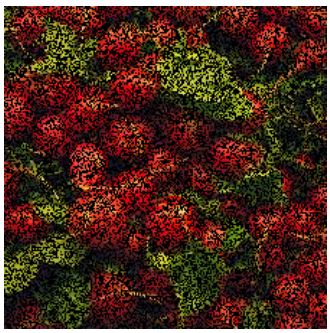


(F) Synthesised colour distribution

FIGURE 4.12: Case 3 applied to radishes. The mask is randomly generated with a probability of 0.5. Mask updated with min pooling.

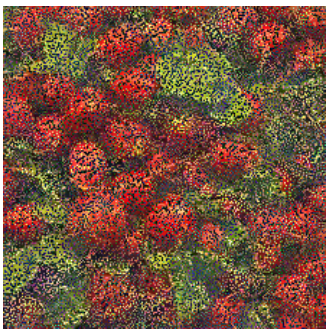
Case 4: masked and constrained. Figure 4.13 contains the results for an unmasked target image with constrained optimisation, given that the mask is updated with *max pooling*.

Figure 4.14 contains the results for an unmasked target image with constrained optimisation, given that the mask is updated with *min pooling*.

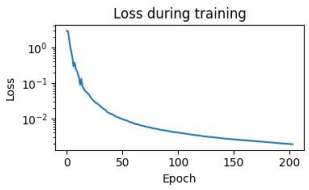


(A) Target

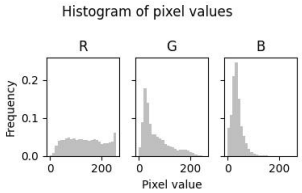
(B) Initial image



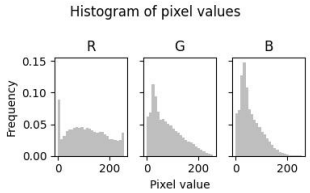
(C) Synthesised



(D) Losses

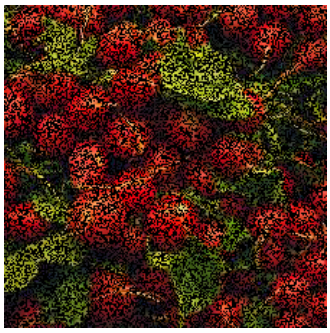


(E) Original colour distribution



(F) Synthesised colour distribution

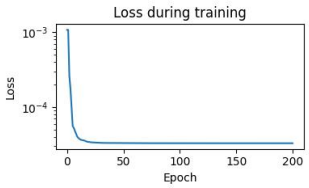
FIGURE 4.13: Case 4 applied to radishes. The mask is randomly generated with a probability of 0.5. Mask updated with max pooling.



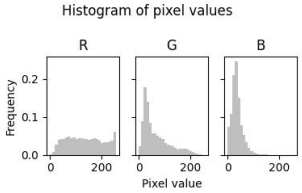
(A) Target

(B) Initial image

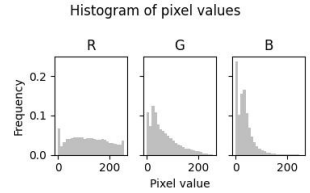
(C) Synthesised



(D) Losses



(E) Original colour distribution



(F) Synthesised colour distribution

FIGURE 4.14: Case 4 applied to radishes. The mask is randomly generated with a probability of 0.5. Mask updated with min pooling.

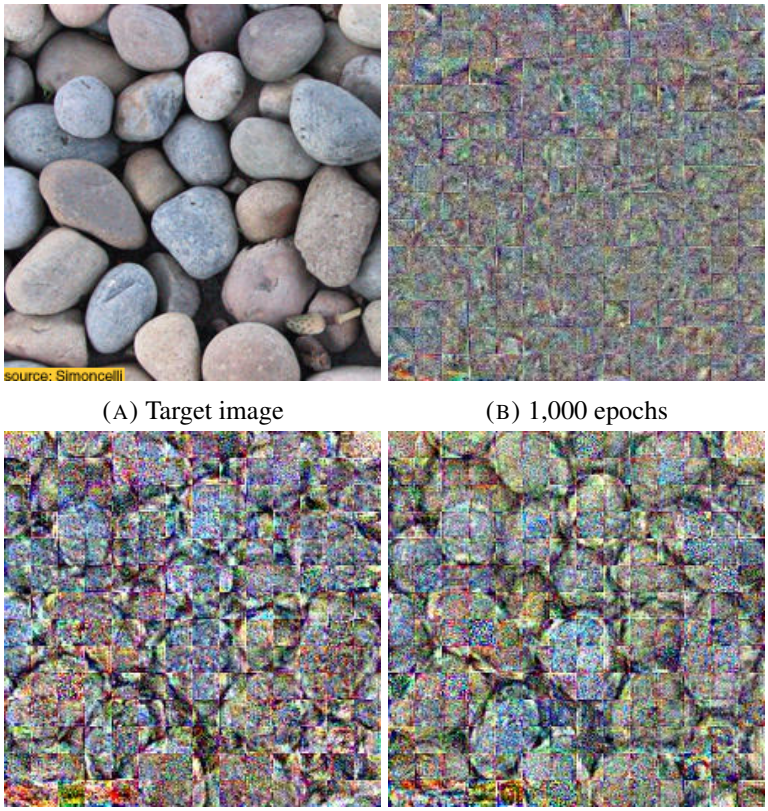
4.2 Structure-aware texture synthesis

We explore two approaches to structure-aware texture synthesis: one based on Vision Transformer and the other based on a gradient loss.

4.2.1 Neural texture synthesis with Vision Transformer

We consider two settings for texture synthesis with Vision Transformer. In the first, texture synthesis is purely based on the Gram matrix representations computed from all layers of a Vision Transformer. In the second, synthesised image resulting from ViT-based texture synthesis is used as a guided initialisation for the neural texture synthesis method by Gatys et al. [6].

Vision Transformer. It can be seen in Figure 4.15 that texture synthesis purely based on ViT has an image intensity gradient that has some semblance to the gradient of the original image's intensity. Nevertheless, the original colour values are not reproduced. Instead, the texture consists of a noisy pattern that is reminiscent of Gaussian noise, albeit with additional local patterns.



(A) Target image (B) 1,000 epochs

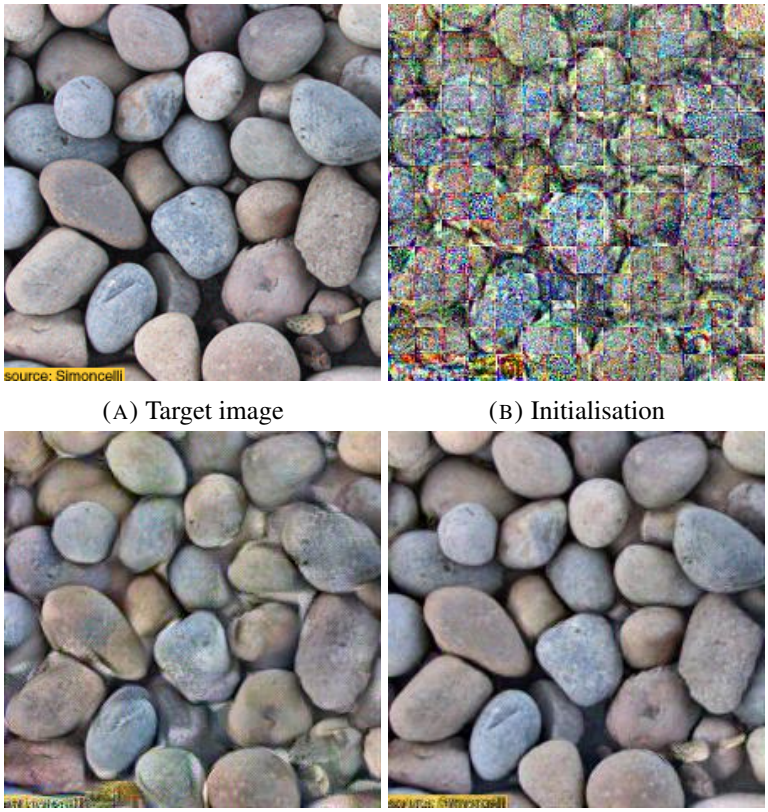
(C) 10,000 epochs (D) 15,000 epochs

FIGURE 4.15: ViT-based texture synthesis of pebbles.

VGG-19 and Vision Transformer. Figure 4.16 shows VGG-19-based texture synthesis with an initialisation that is not based on Gaussian noise, but which is instead the result of ViT-based texture synthesis. The mean squared error between the original image and the synthesised image is 293.1 and the mean absolute error is 13.4.

Figure 4.17 shows the same procedure applied to an image of cans. With this target image, the mean squared error is higher at 4167.6. The mean absolute error is 49.9, indicating almost a four-fold increase in error with respect to the procedure applied to the pebbles texture. Figure 4.17 shows a decreased capacity of the consecutive

application of VGG-19-based and ViT-based texture synthesis to accurately capture the global structure, compared to the previous figure.



(A) Target image (B) Initialisation

(C) 250 epochs (D) 2,500 epochs

FIGURE 4.16: VGG-19 based texture synthesis with a ViT-guided texture initialisation on a texture of pebbles.

(A) Target image (B) Initialisation

(C) 250 epochs (D) 2,500 epochs

FIGURE 4.17: VGG-19 based texture synthesis with a ViT-guided texture initialisation on a texture of stacked cans.

4.2.2 Neural texture synthesis with a gradient loss

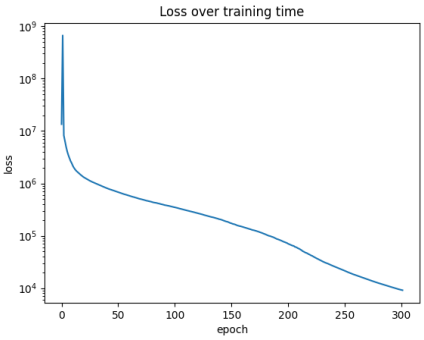
Figure 4.18 shows the results of texture synthesis including a gradient-based loss on an image of a cat. The appendix contains the application of this method to various images of textures. The results were created with $\alpha = 10^7$.

The mean squared error between the target image in Figure 4.18a and the synthesised image in Figure 4.18d is 729.0, while its mean absolute error is 21.2.

(A) Target image

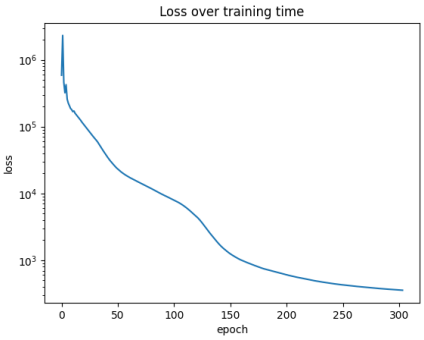
(B) $L = L_{Gatys}$

(C) $L = L_{Gatys} + L_{grad}$



(D) Loss $L_{Gatys} + L_{grad}$

(E) $L = L_{grad}$



(F) Loss L_{grad}

(G) $G_x(T)$

(H) $G_y(T)$

FIGURE 4.18: A gradient loss combined with the Gram matrix representation-based loss proposed by Gatys et al. applied to an image of a cat.

5 Discussion

5.1 Mask-aware texture synthesis

5.1.1 Case 1: unmasked and unconstrained

The short-range textural structures are captured by the mask-aware neural network and show many similarities to the short-range textural structures exhibited in Figure 2.7b. The difference in mask modification is most obvious at pixels proximate to the edges, where a min pooling strategy causes the patterns to blur. This is as expected, since the min pooling operation causes the mask to shrink over an increased depth in the neural network. In other words, the mask-aware network with a min pooling mask update method does not 'perceive' boundary pixels in layers above the first mask-aware convolutional layer.

5.1.2 Case 2: unmasked and constrained

In case 2, the synthesised texture comes close to the global minimum that is the original texture with max pooling method, as indicated by the root mean squared error of below 8 and the mean absolute error of 1.6 for the radishes image. Details like white endings of radishes and specific irregularities in the leaves are reproduced.

With the substitution of the max pooling strategy by the min pooling strategy, there is a change in textural structure and colour distribution. The algorithm does not converge to the global minimum. Instead, the synthesised pixels are lighter than the constrained ones.

5.1.3 Case 3: masked and unconstrained

Case 3 with max pooling exhibits straight, dark lines as part of the synthesised texture. This is likely due to the sensitivity of the algorithm to boundaries, which are black due to the Hadamard product of the mask and the input. Indeed, min pooling does not contain dark lines. This advantage is counterbalanced by the blurring of textural structures near the border.

5.1.4 Case 4: masked and constrained

Case 4 performs best when min pooling is used to update the masks. When max pooling is used instead, the network likely reproduces the black boundaries it detects in the original texture within the area that it needs to synthesise.

With a min pooling mask update strategy, the colour distribution displays a slight rightward shift, adding the requirement of post-processing steps like histogram matching to the synthesised area specifically. However, low-level textural structures emerge that bear similarities to the original texture in the same way as the original method proposed by Gatys et al. [6]. Moreover, the synthesised pixels show a slight blurring, which was also present in the min pooling version of case 2 and 3. Unfortunately, though, the max pooling method, which performs better for cases

If this method were used as an inpainting algorithm, its notable strength would lie in its ability to function without the need for training. As a result, the need for large datasets or vast computational resources is eliminated and substantial computational, making the inpainting algorithm accessible to a broader audience. The model's independence from a vast dataset also mitigates concerns about data cleanliness and potential artifacts, affording users increased control and transparency in the inpainting process. However, it is essential to acknowledge its limitations, as the method still needs post-processing steps to improve the colour distribution. Moreover, the method is specifically intended for extending textures and lacks the capability to comprehend higher-level structures. Additionally, while the inference phase may demand more resources, this is counterbalanced by the reduced upfront resource requirements. Ultimately, the decision on whether to employ this inpainting method depends on the specific use case, allowing users to weigh the advantages and limitations based on their unique situation.

The method could be extended to function in a two-stream architecture separating a texture and a structure stream, as a similar but alternative method to the texture and structure dual generation inpainting method proposed by Guo et al. [8]. Their method is based on the alternative hourglass-shaped U-Net-architecture and needs to be trained on extensive datasets, instead of a pyramid-shaped neural network architecture like VGG-19 which needs no training.

5.2 Structure-aware texture synthesis

5.2.1 Neural texture synthesis with Vision Transformer

A Gram matrix representation is not a static texture representation of ViT activations that captures the full texture characteristics, even though the obtained higher-level structures could potentially be a useful structure extractor. It is however very likely that less computationally intensive methods exist for this purpose, that do the job just as well if not better. Structures that are due to image intensities seem to be more likely to be captured by the ViT-based method than structures based on colour differences, but further research is needed to conclude this with certainty.

By patchifying the original image into very small patches, it is easier to exactly replicate the original structures. Moreover, the current implementation does not allow for upscaling, due to the patches and positional encoding.

5.2.2 Neural texture synthesis with a gradient loss

Figures generated solely based on the gradient loss show that this loss function captures a lot of information of relatively detailed textures, despite the presence of blurring in the computation of the gradient loss. With increased blurring, however, some of the larger structures were not preserved in the synthesised images, such as the position of the eyes in the cat image.

In Figure B.3, we can clearly see the influence of image intensity on gradient-based texture synthesis. Here, the structure is influenced by the occurrence of a shadow, that distorts the gradients of the texture, even though to the human eye texture perception would not depend on a sporadic shadow.

Though the method is simple, it works relatively well to preserve structure. The main area of improvement is the exclusion of relative differences in image intensity that are relatively large, not excluded by blurring and part of the texture. Currently, disrepancies between the original and synthesised images are largely due to colour dissimilarities, but excluding this larger structures that are actually part of the texture from the structure loss could increase the uniqueness of the synthesised image.

6 | Conclusion

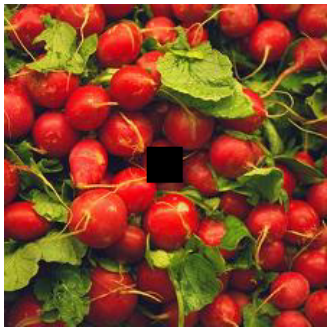
This report proposed a mask-aware extension to the neural texture synthesis method first proposed by Gatys et al. [6]. Though various settings show different performances, the mask-aware VGG-19 network is an effective mask-aware generalization of the original algorithm. Textural structures are synthesised, though slightly blurrier versions when a min pooling method is used to perpetuate the masks through the network layers. The method could, however, be an interesting alternative to U-Net-based inpainting methods, as it requires no additional training, especially when used in conjunction with pretrained texture segmentation networks.

In addition, this report researched the suitability of applying the concept of Gram matrix-based texture synthesis to a Vision Transformer, due to its inductive bias of globality. Though this method captures long-range textural structures, its texture is not synthesised accurately. When using this ViT-based texture synthesis as a structure prior for the original CNN-based neural texture synthesis method, the original image can be reproduced with only a small mean squared error.

The final research focus was neural texture synthesis with a gradient loss, to enable structure-aware texture synthesis. Though the method is simple, it works relatively well to preserve structure. The synthesised image mainly varies in colour, however, so it could be an interesting direction of future investigation to reduce the amount of small textural structure that is captured in the structure loss.

A | Additional results - mask-aware texture synthesis

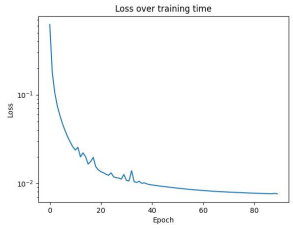
Figure A.1 shows a result of case 3 with max pooling to update the masks. It can clearly be seen that the algorithm considers the black boundaries of the hole as part of the texture, as a black square is synthesised in another position than the masked area in the original mask.



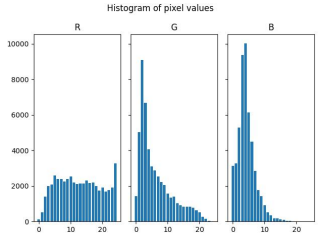
(A) Target

(B) Initial image

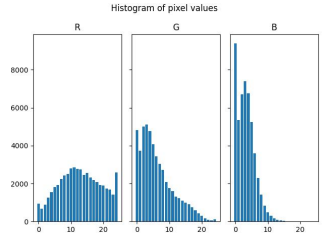
(C) Synthesised after 90 epochs



(D) Losses



(E) Original colour distribution



(F) Synthesised colour distribution

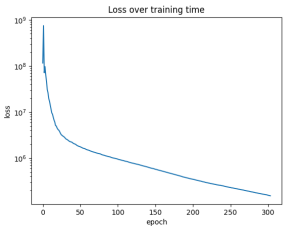
FIGURE A.1: Case 3 applied to radishes. Mask updated with max pooling.

B | Additional results - structure-aware texture synthesis

(A) Target image

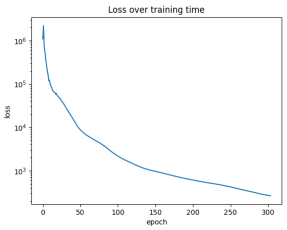
(B) $L = L_{Gatys}$

(C) $L = L_{Gatys} + L_{grad}$

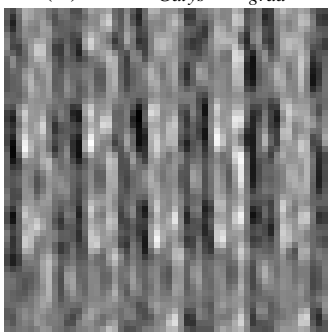


(D) Loss $L_{Gatys} + L_{grad}$

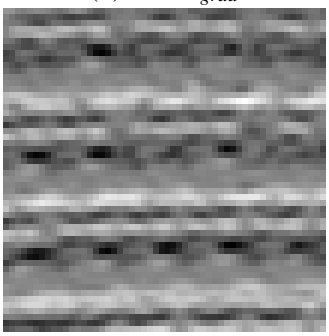
(E) $L = L_{grad}$



(F) Loss L_{grad}



(G) $G_x(T)$



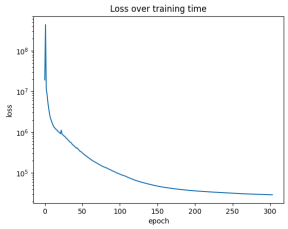
(H) $G_y(T)$

FIGURE B.1: A gradient loss combined with the Gram matrix representation-based loss proposed by Gatys et al. applied to an image of cans.

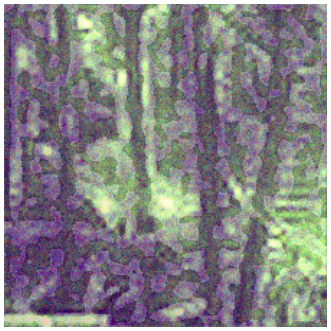
(A) Target image

(B) $L = L_{Gatys}$

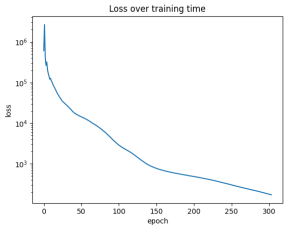
(C) $L = L_{Gatys} + L_{grad}$



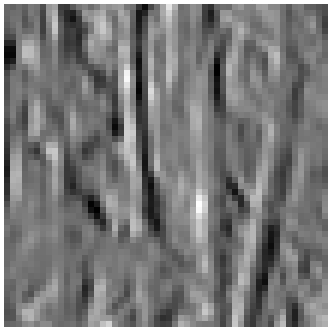
(D) Loss $L_{Gatys} + L_{grad}$



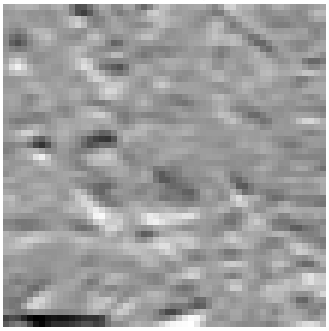
(E) $L = L_{grad}$



(F) Loss L_{grad}



(G) $G_x(T)$



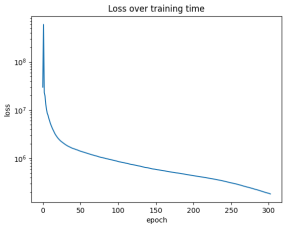
(H) $G_y(T)$

FIGURE B.2: A gradient loss combined with the Gram matrix representation-based loss proposed by Gatys et al. applied to an image of jungle.

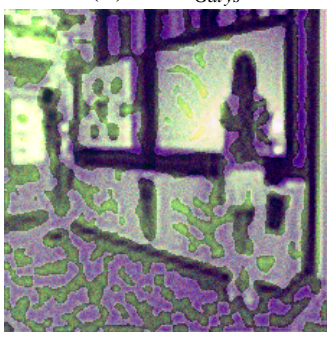
(A) Target image

(B) $L = L_{Gatys}$

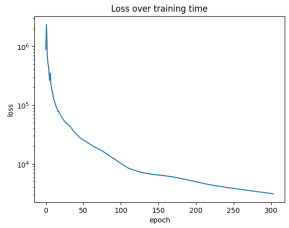
(C) $L = L_{Gatys} + L_{grad}$



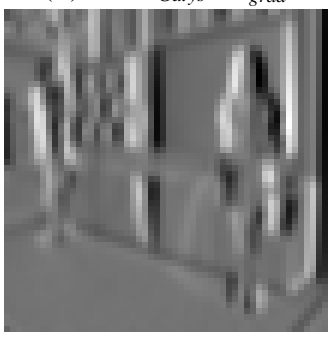
(D) Loss $L_{Gatys} + L_{grad}$



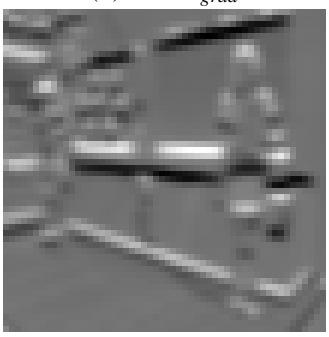
(E) $L = L_{grad}$



(F) Loss L_{grad}



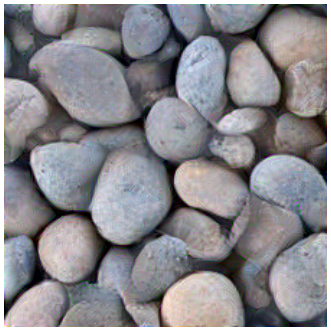
(G) $G_x(T)$



(H) $G_y(T)$

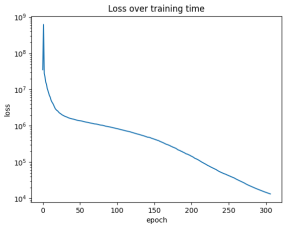
FIGURE B.3: A gradient loss combined with the Gram matrix representation-based loss proposed by Gatys et al. applied to an image of the inside of a shop.

(A) Target image

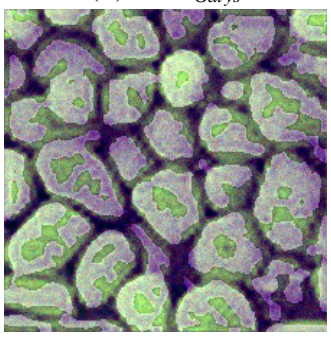


(B) $L = L_{Gatys}$

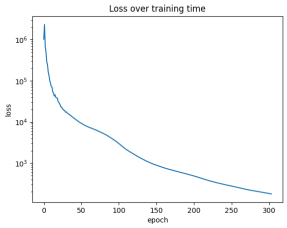
(C) $L = L_{Gatys} + L_{grad}$



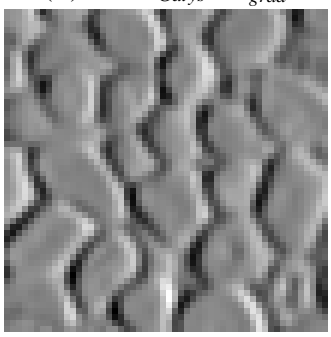
(D) Loss $L_{Gatys} + L_{grad}$



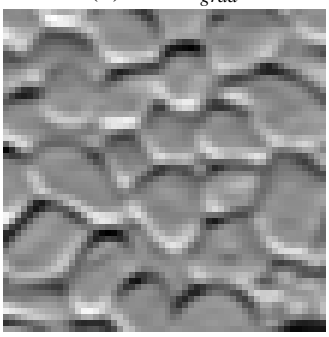
(E) $L = L_{grad}$



(F) Loss L_{grad}



(G) $G_x(T)$



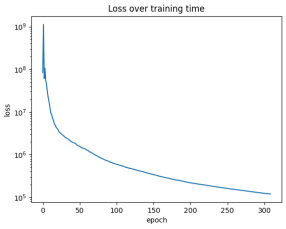
(H) $G_y(T)$

FIGURE B.4: A gradient loss combined with the Gram matrix representation-based loss proposed by Gatys et al. applied to an image of pebbles.

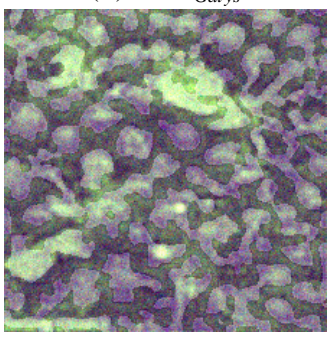
(A) Target image

(B) $L = L_{Gatys}$

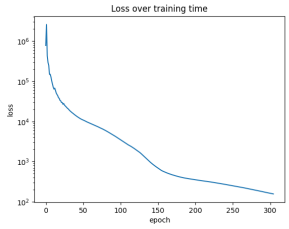
(C) $L = L_{Gatys} + L_{grad}$



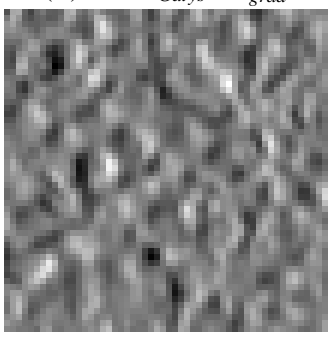
(D) Loss $L_{Gatys} + L_{grad}$



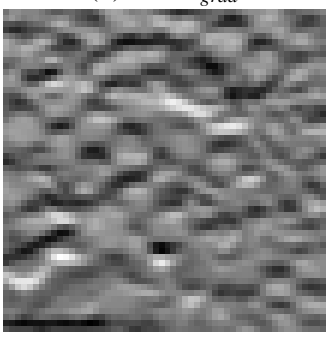
(E) $L = L_{grad}$



(F) Loss L_{grad}



(G) $G_x(T)$



(H) $G_y(T)$

FIGURE B.5: A gradient loss combined with the Gram matrix representation-based loss proposed by Gatys et al. applied to an image of radishes.

Bibliography

[1] A. Akl, C. Yaacoub, M. Donias, J.-P. Da Costa, and C. Germain. A survey of exemplar-based texture synthesis methods. *Computer Vision and Image Understanding*, 172:12–24, 2018.

[2] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.

[3] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3606–3613, 2014.

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[5] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. *ICCV*, 1999.

[6] L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. *Advances in neural information processing systems*, 28, 2015.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[8] X. Guo, H. Yang, and D. Huang. Image inpainting via conditional texture and structure dual generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14134–14143, 2021.

[9] E. Heitz, K. Vanhoey, T. Chambon, and L. Belcour. A sliced wasserstein loss for neural texture synthesis. In *CVPR*, pages 9412–9420. 2021.

[10] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 33:6840–6851, 2020.

[11] E. Hüllermeier, T. Fober, and M. Mernberger. *Inductive Bias*, pages 1018–1018. Springer New York, New York, NY, 2013.

[12] N. Jetchev, U. Bergmann, and R. Vollgraf. Texture synthesis with spatial generative adversarial networks. *NeurIPS*, 2016.

[13] B. Julesz. Visual pattern discrimination. *IRE transactions on Information Theory*, 8(2):84–92, 1962.

[14] B. Julesz, E. N. Gilbert, L. A. Shepp, and H. L. Frisch. Inability of humans to discriminate between visual textures that agree in second-order statistics—revisited. *Perception*, 2(4):391–405, 1973.

[15] W.-C. Lin, J. Hays, C. Wu, Y. Liu, and V. Kwatra. Quantitative evaluation of near regular texture synthesis algorithms. volume 1, pages 427– 434, 07 2006.

[16] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European conference on computer vision (ECCV)*, pages 85–100, 2018.

[17] T. M. Mitchell. The need for biases in learning generalizations. 1980.

[18] J. Portilla and E. P. Simoncelli. Representation and synthesis of visual texture. `https://www.cns.nyu.edu/~lcv/texture/`. Accessed: 2023-11-08.

[19] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 40:49–70, 2000.

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

[21] M. L. Richter, J. Schöning, A. Wiedenroth, and U. Krumnack. Receptive field analysis for optimizing convolutional neural network architectures without training. In *Deep Learning Applications, Volume 4*, pages 235–261. Springer, 2022.

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

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[24] E. P. Simoncelli and W. T. Freeman. The steerable pyramid: A flexible architecture for multi-scale derivative computation. `https://www.cns.nyu.edu/~eero/steerpyr/`, 1995. Accessed: 2023-11-08.

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR 2015)*. Computational and Biological Learning Society, 2015.

[26] L. Surace, M. Wernikowski, C. Tursun, K. Myszkowski, R. Mantiuk, and P. Didyk. Learning gan-based foveated reconstruction to recover perceptually important image features. *ACM Transactions on Applied Perception*, 20(2), 2023.

[27] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep image prior. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9446–9454, 2018.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *NeurIPS*, 30, 2017.