HW3

Gabriel ROMON

1 Texture synthesis a la Gatys

1. The VGG-19 architecture is shown in Figure 1. In [1] Gatys uses convolutional layers of this network to compute Gram matrices G^l that act as stationary descriptions of the input texture. In the code provided for the practical session the line
   `TEXTURE_LAYERS = ['conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv5_1']`
   selects the layers for which Gram matrices are computed. In this case, only the first layer of each block is selected.

   With this choice of layers, results for the pebble texture with 2000 epochs are shown in Figure 2. The results are quite satisfactory although the shape of the pebbles is sometimes not quite round. It is interesting to note that the yellow watermark in the original texture has been replicated in the bottom right of the generated picture.

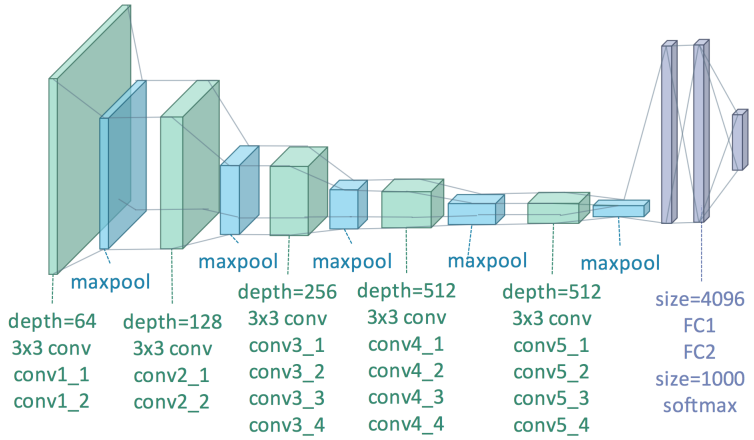

Figure 1: VGG-19 architecture

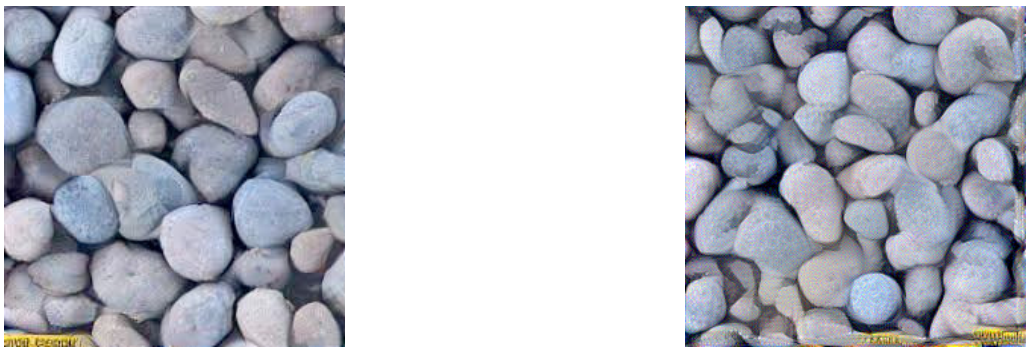

Figure 2: (Left) Original pebble texture (Right) Texture synthesised after 2000 epochs, Gram matrices from the first layer of each block

2. I experimented by progressively adding deeper layers in Figure 3: image (a) uses only the Gram matrix from the first layer of the first block, image (b) uses this layer as well as the first layer of the second block, and so on. The number of epochs was set to 2000 in each case. Using only layers from the first three blocks does not yield convincing textures (images (a) to (c)). However, with 4 blocks the texture is satisfying and I do not notice any significant improvement when adding the layer from the fifth block.

   In Figure 4 I selected only the layers of the last block. While the round outlines of pebbles are visible, low-level details have been lost. Note that the last block comes after 4 max-pooling layers so the receptive field of the corresponding convolution layers is quite large, which explains why low-level details are gone.

   In Figure 5 I added all the layers from all the blocks. This does not improve the texture compared to Figure 2.

(a) 1 layer (b) 2 layers (c) 3 layers (d) 4 layers (e) 5 layers

Figure 3: Effect of adding the first layer of each block (from left to right)

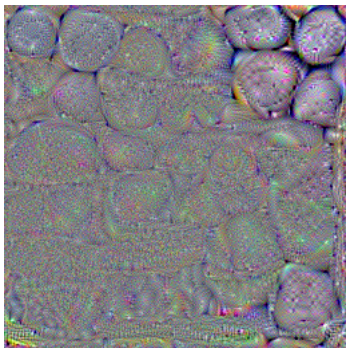

Figure 4: Effect of selecting only the layers of the last block

Figure 5: Effect of selecting all the layers from all the blocks

3. The optimizer used in the code of the practical session is Adam. In [1] the authors use L-BFGS instead, reportedly because it is more suited to the high-dimensional setting of the problem.

2 Texture synthesis with TextureNet

1. Since the generator network is only trained to minimize the texture loss

$$\mathcal{L}_T\left(\mathbf{x};\mathbf{x}_0\right)=\sum_{l\in L_T}\left\|G^l(\mathbf{x})-G^l\left(\mathbf{x}_0\right)\right\|_2^2$$

it may generate textures that have good quality but very low diversity (different input noises $\mathbf{z}$ produce similar textures). This issue is similar to mode collapse with GANs. In [2] the authors provide the example shown in Figure 6 of a generator that overfits and produces textures that have very similar structures.

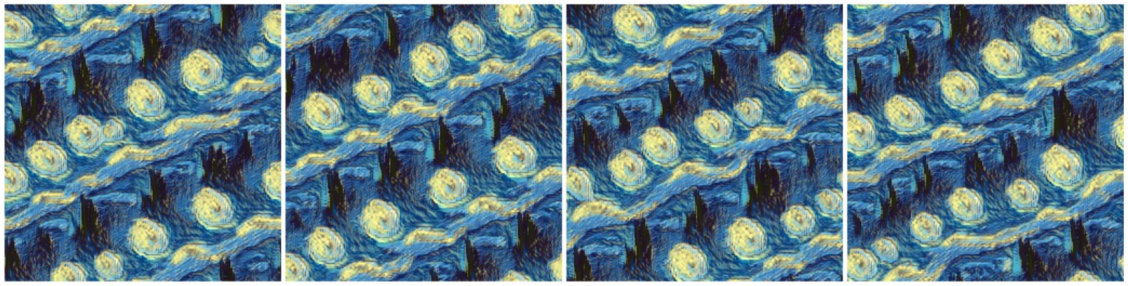

Figure 6: Four textures produced by a generator that overfits

2. The architecture of the generator network is shown in Figure 7. Ulyanov developed a multi-scale architecture: the input noise is actually made up of 5 random subnoises that have sizes $\frac{256}{2^i}\times\frac{256}{2^i}$ for i in $\{1,\ldots,5\}$. As stated in [2], "Each random noise tensor is first processed by a sequence of convolutional and non-linear activation layers, then upsampled by a factor of two, and finally concatenated as additional feature channels to the partially processed tensor from the scale below". In the paper, the authors specify that the model has ~ 65.000 parameters.

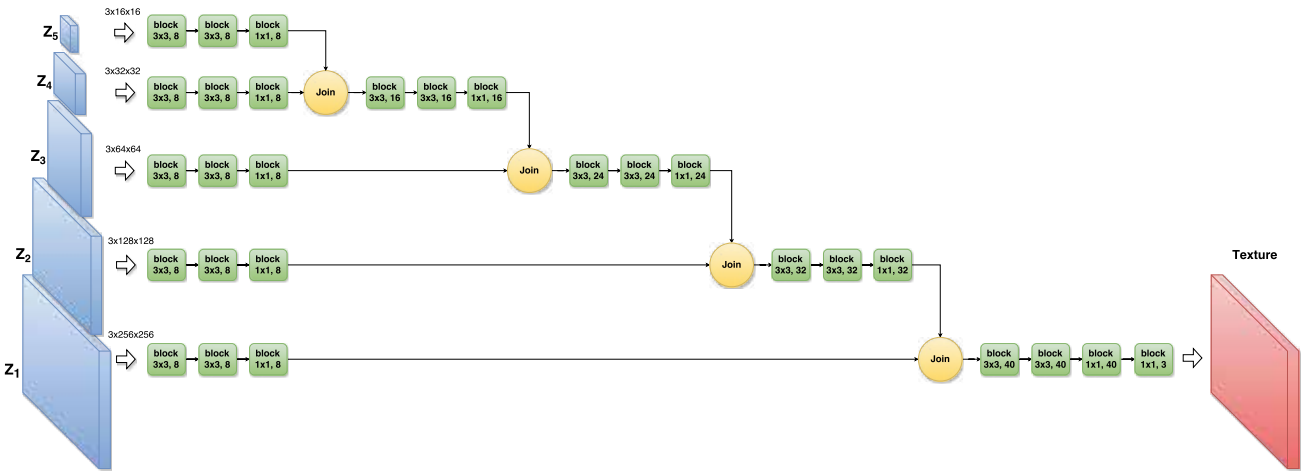

Figure 7: Architecture used for the generator network

3. Results generated by the network for the brick, lego and metal texture are shown in Figure 8. I sampled 5 random noises for each category and used the corresponding pretrained network to generate new textures. Since the practical session relies on pretrained models, the time and ressources needed to produce textures with this technique are very low.
   The results for brick and metal are satisfying: low-level and high-level details have been preserved. However, there might be a lack of diversity in the brick textures (some of them look quite similar). Performance for the lego texture is very poor nonetheless.

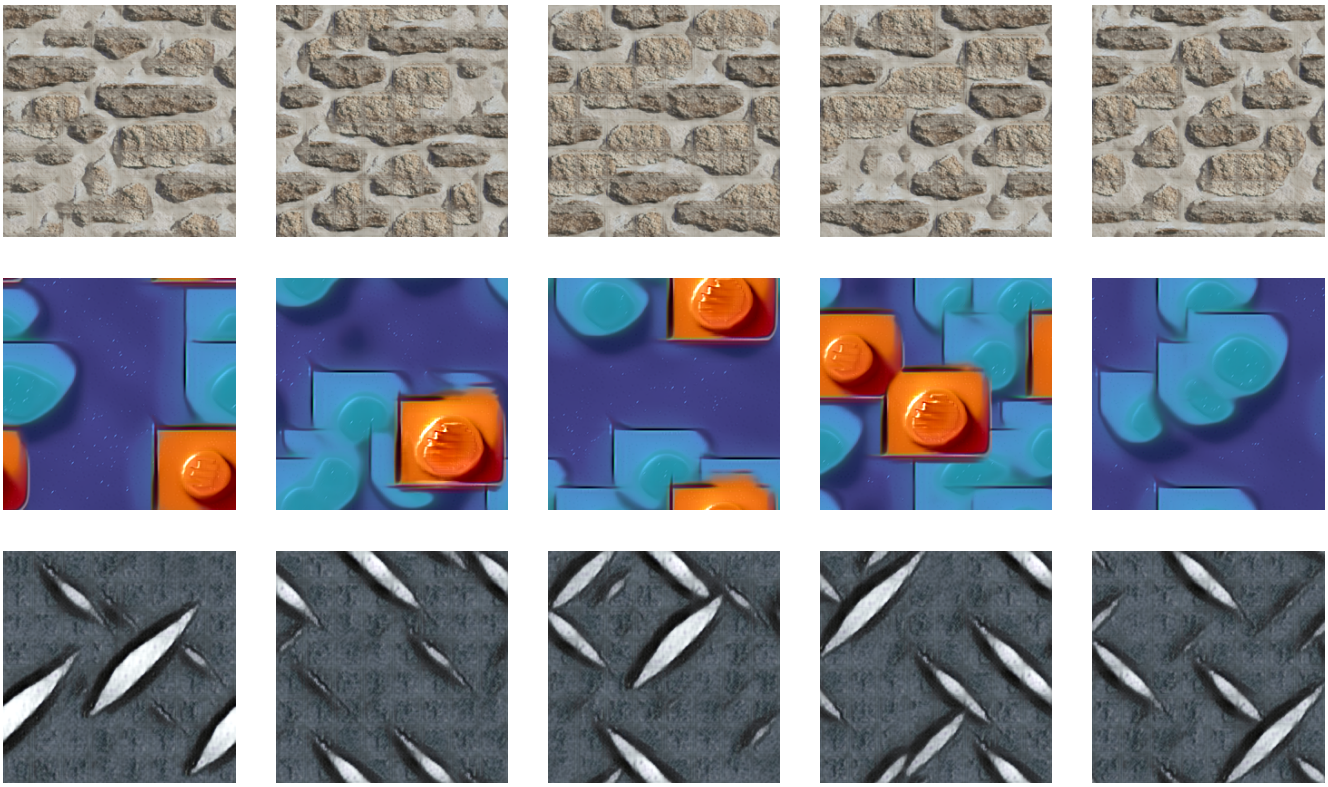

Figure 8: Textures generated by the network

4. Linear interpolation between two noises is achieved with the instruction
   `k[t,:,:,:]=lam*k[0,:,:,:] + (1-lam)*k[-1,:,:,:]`

5. Textures at the quarter of the sequence tend to be low-quality. This is because of the linear interpolation on the noise: a convex combination of two uniform distributions need not be uniform, so the interpolated noises may have very small probability of being samples from a uniform distribution, hence the poor textures.

References

[1] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in neural information processing systems*, pages 262–270, 2015.

[2] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, volume 1, page 4, 2016.