Beyond GOTEX: Using multiple feature detectors for better texture synthesis

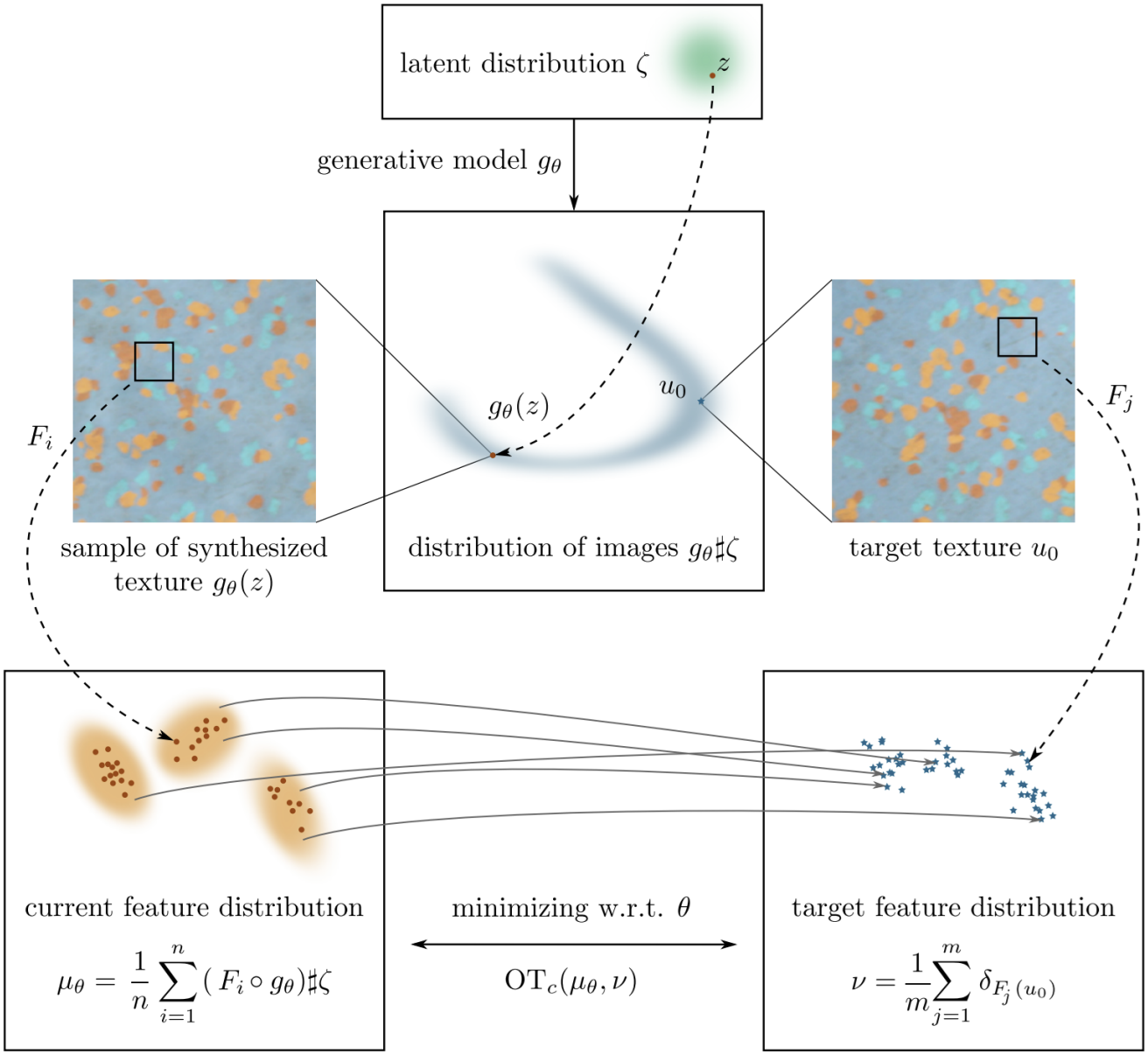
Project presentation: A Generative Model for Texture Synthesis based on Optimal Transport Between Feature Distribution

Alessio Spagnoletti

ENS Paris-Saclay

April 11, 2024

GOTEX framework



GOTEX loss function

The whole idea of the GOTEX framework is to use an OT cost function as loss since we are working with probabilities:

$$\mathcal{L}_{\text{GOTEX}}(\theta, u_0) = \text{OT}_c(\mu_\theta, \nu) = \inf_{\pi \in \Pi(\mu_\theta, \nu)} \int c(x, y) d\pi(x, y).$$

Thanks to the semi-dual and semi-discrete formulation, we get a min-max problem:

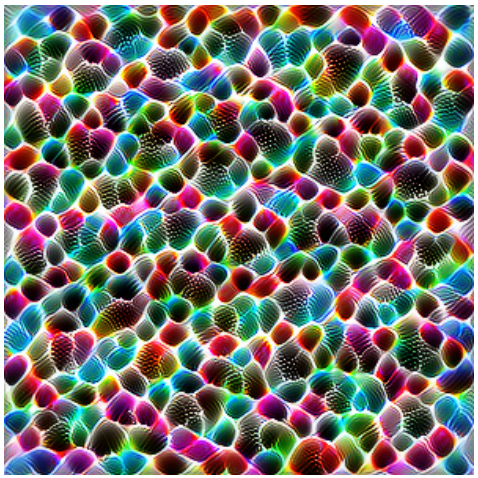
$$\begin{aligned}
&\min_\theta \mathcal{L}_{\text{GOTEX}}(\theta, u_0) = \min_\theta \text{OT}_c(\mu_\theta, \nu) = \\
&= \min_\theta \sup_{\psi \in L^1(\nu)} \mathcal{J}(\theta, \psi) := \int \psi^c(x) d\mu_\theta(x) + \int \psi(y) d\nu(y) = \\
&= \min_\theta \max_{\psi \in \mathbb{R}^m} \mathbb{E}_{Z \sim \zeta} \left[\frac{1}{n} \sum_{i=1}^n \psi^c(F_i \circ g_\theta(Z)) + \frac{1}{m} \sum_{j=1}^m \psi_j \right]
\end{aligned}$$

where the inner max problem can be proved to be concave

Our main contribution

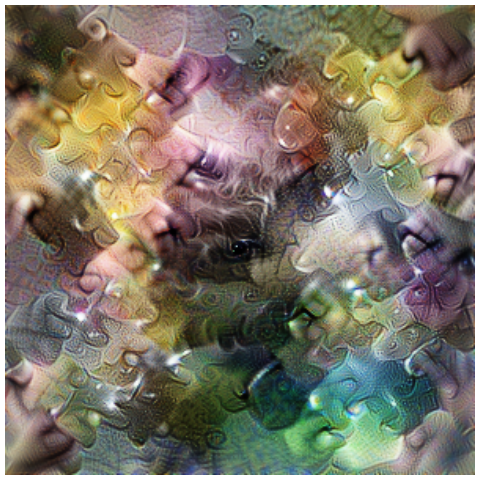
The authors of GOTEX decided to use Gaussian patches and Deep features from the VGG-19 network as the main features extractor. We also tried the same approach but with the deep features extracted from the Inception V3 NN.

We thought about them after looking at the optimized images that maximize a given unit's activation (a specific convolutional layer filter).



ch. 48 - layer 5d

ch. 5 - layer 6e



ch. 73 - layer 7c

If we compare them to their VGG-19 counterparts, we find that many patterns are better recognized, also due to the larger number of filters used.

The losses

Since we will deal with many features, we will consider several different losses:

$$\mathcal{L}_{\text{patch}}(\theta) = \sum_{l=1}^{N_F} \mathrm{OT}_c\left(\mu^l_{\text{pat}}(\theta), \nu^l_{\text{pat}}\right)$$

$$\mathcal{L}_{\mathrm{VGG}}(\theta) = \sum_{l=1}^{N_F} \mathrm{OT}_c\left(\mu^l_{\mathrm{VGG}}(\theta), \nu^l_{\mathrm{VGG}}\right)$$

$$\mathcal{L}_{\mathrm{all}}(\theta) = \lambda \sum_{l=1}^{N_F} \mathrm{OT}_c\left(\mu^l_{\mathrm{pat}}(\theta), \nu^l_{\mathrm{pat}}\right) + \sum_{l=1}^{N_F} \mathrm{OT}_c\left(\mu^l_{\mathrm{VGG}}(\theta), \nu^l_{\mathrm{VGG}}\right)$$

$$\mathcal{L}_{\mathrm{IV3}}(\theta) = \sum_{l=1}^{N_F} \mathrm{OT}_c\left(\mu^l_{\mathrm{pat}}(\theta), \nu^l_{\mathrm{pat}}\right) + \lambda \sum_{l=1}^{3} \mathrm{OT}_c\left(\mu^l_{\mathrm{IV3}}(\theta), \nu^l_{\mathrm{IV3}}\right)$$

A code *morceau*

```
for it in range(iter_max):

    # 1. update psi

    for itp in range(iter_psi):
        synth_features = [A for _ , A in FeatExtractor(synth_img).items()] # evaluate on the current synthetized image
        for i, feat in enumerate(synth_features):
            psi_optimizers[i].zero_grad()
            loss = -ot_layers[i](feat.detach())
            loss.backward()
            # Normalize gradient
            ot_layers[i].dualVariablePsi.grad.data /= ot_layers[i].dualVariablePsi.grad.data.norm()
            psi_optimizers[i].step()
            # Set psi as the average of the parameters, as stored in the optim's buffer 'ax'
            ot_layers[i].dualVariablePsi.data = psi_optimizers[i].state[ot_layers[i].dualVariablePsi]['ax'] if itp==iter_psi-1

    input_downsampler(synth_img.detach()) # evaluate on the current synthetized image
    for s in range(n_scales):
        optim_psi = torch.optim.ASGD([semidual_loss[s].dualVariablePsi], lr=1, foreach=False, alpha=0.5, t0=1)
        for i in range(iter_psi):
            fake_data = input_im2pat(input_downsampler[s].down_img, -1)
            optim_psi.zero_grad()
            loss = -semidual_loss[s](fake_data)
            loss.backward()
            # Normalize gradient
            semidual_loss[s].dualVariablePsi.grad.data /= semidual_loss[s].dualVariablePsi.grad.data.norm()
            optim_psi.step()
        # Set psi as the average of the parameters, as stored in the optim's buffer 'ax'
        semidual_loss[s].dualVariablePsi.data = optim_psi.state[semidual_loss[s].dualVariablePsi]['ax']
```

Dual variable Psi update

A code *morceau*

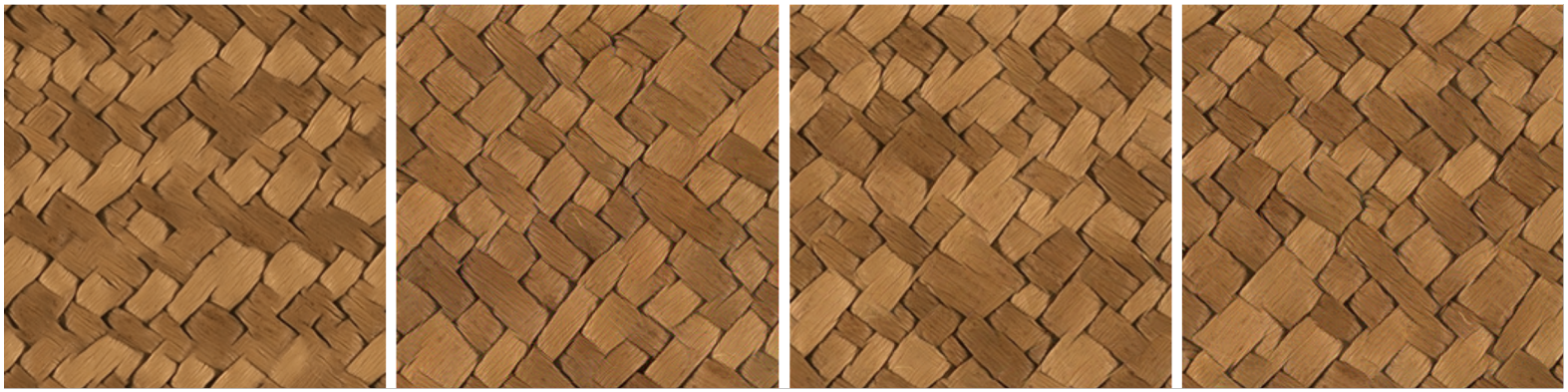
```
# 2. perform gradient step on the image
image_optimizer.zero_grad()
tloss = 0

for s in range(n_scales):
    input_downsampler(synth_img)
    fake_data = input_im2pat(input_downsampler[s].down_img, -1)
    loss = prop[s]*semidual_loss[s](fake_data)
    loss.backward()
    tloss += loss.item()

for i in range(n_layers):
    synth_features = [A for _ , A in FeatExtractor(synth_img).items()]
    feat = synth_features[i]
    loss = 0.05*ot_layers[i](feat)
    loss.backward()
    tloss += loss.item()

image_optimizer.step()
```

Image pixels update

Code available at: GitHub rep

Experiments: Wicker test (single image)

The wicker texture was deeply studied in the original paper. We aim to show how our **Gotex-mix-IV3** model performs better than all the others both in the single image case and the generative one.

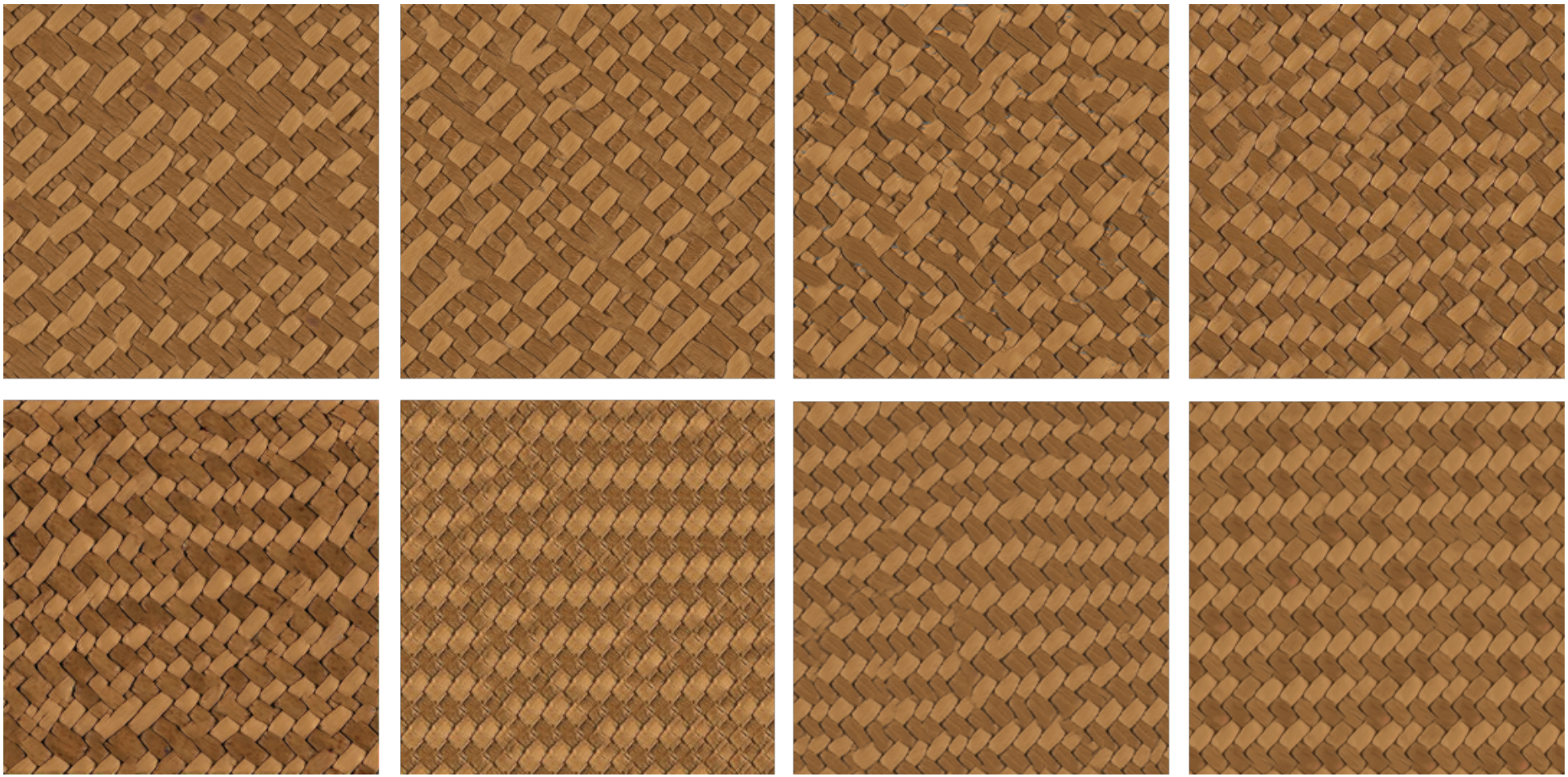


Original tests: Gotex-patch, Gotex-VGG, Gotex-mix, Gotex-all

Real one

Gauss+IV3

Experiments: Wicker test (generative case)



From the up-left: Gotex-mix, Gotex-VGG, SW-VGG, TexNet-VGG, SinGAN (patch + Adv.), PSGAN (patch + Adv.), Gotex-all, Gotex-mix-IV3

Experiments: Wall test

This test is a more complex version of the wicker pattern, featuring diverse bricks in terms of size, shape, and color. We evaluated the top 3 competitors from the previous test.

- Gotex-VGG excels in capturing brick boundaries
- Gotex-IV3 captures color distribution effectively, likely due to Gaussian contributions
- Gotex-all (Gaussian+VGG) performs suboptimally in both aspects

Real one

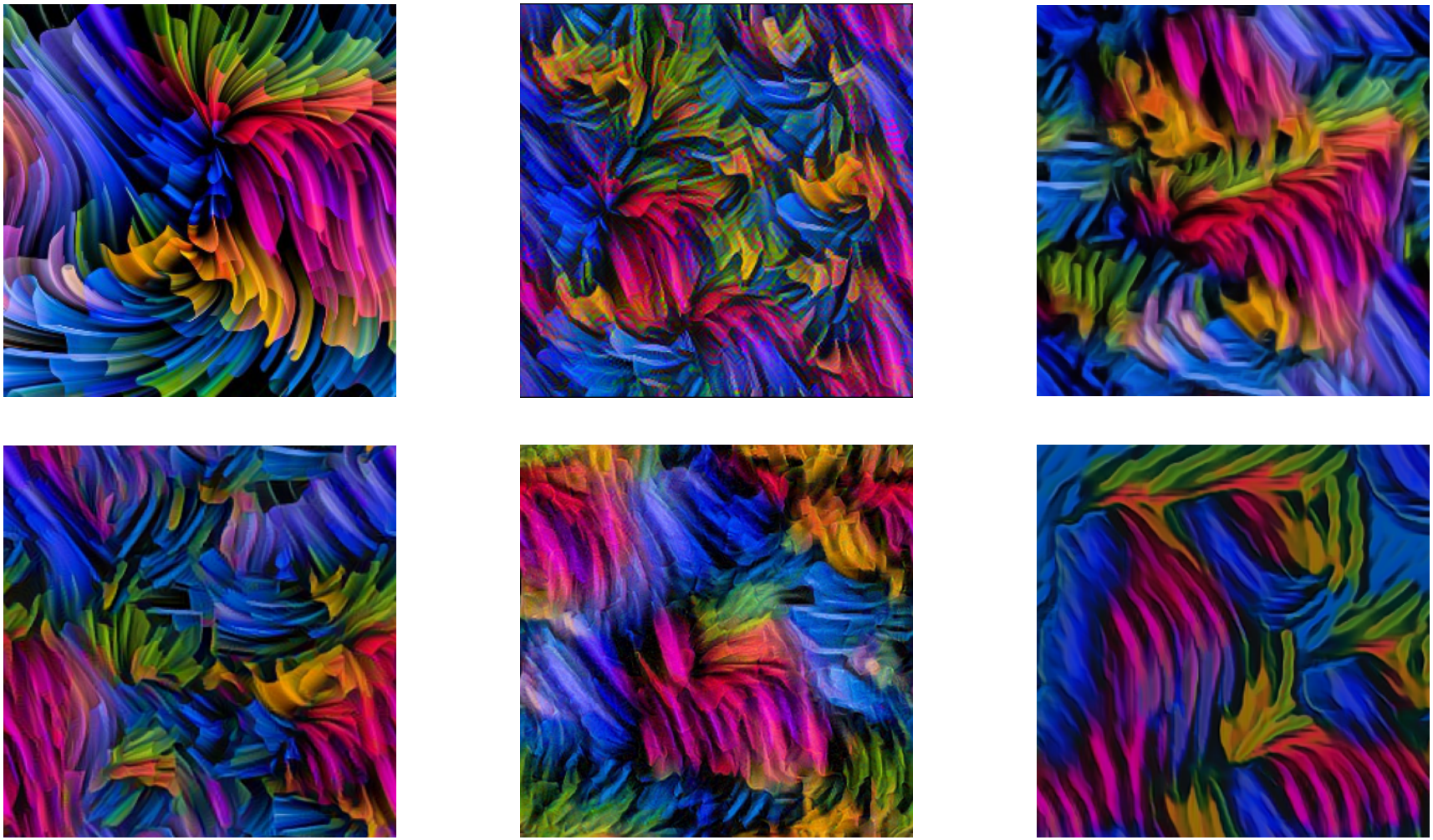
VGG

Gauss+VGG

Gauss+IV3

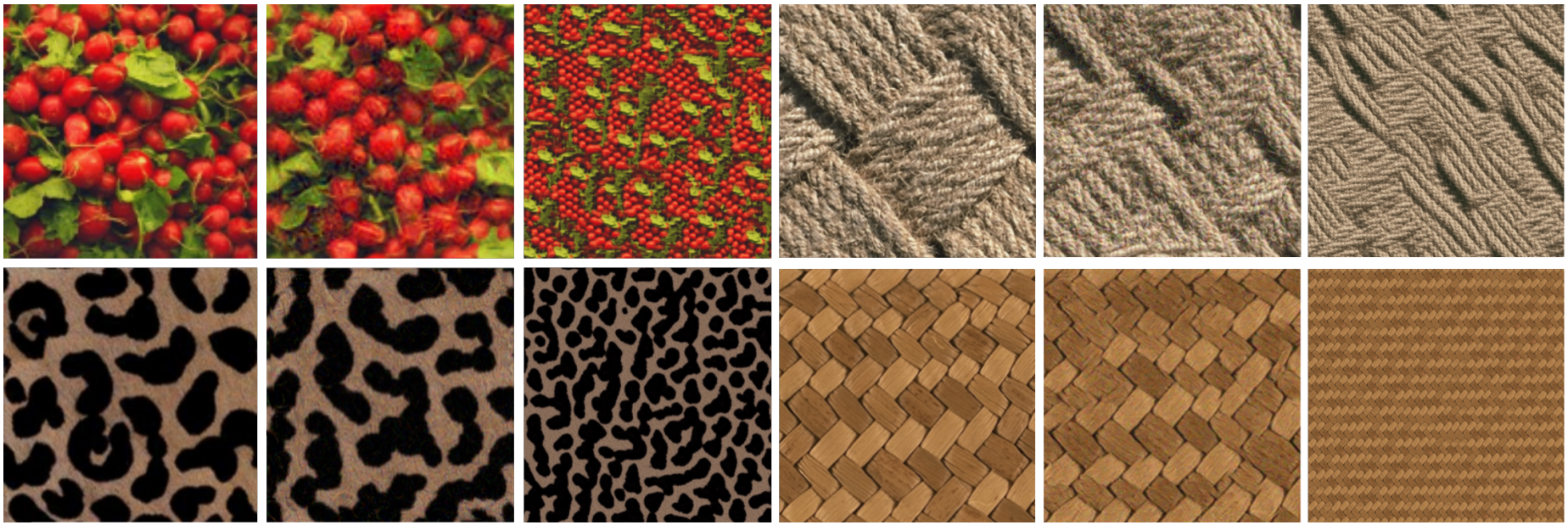
Experiments: Whirlpool test

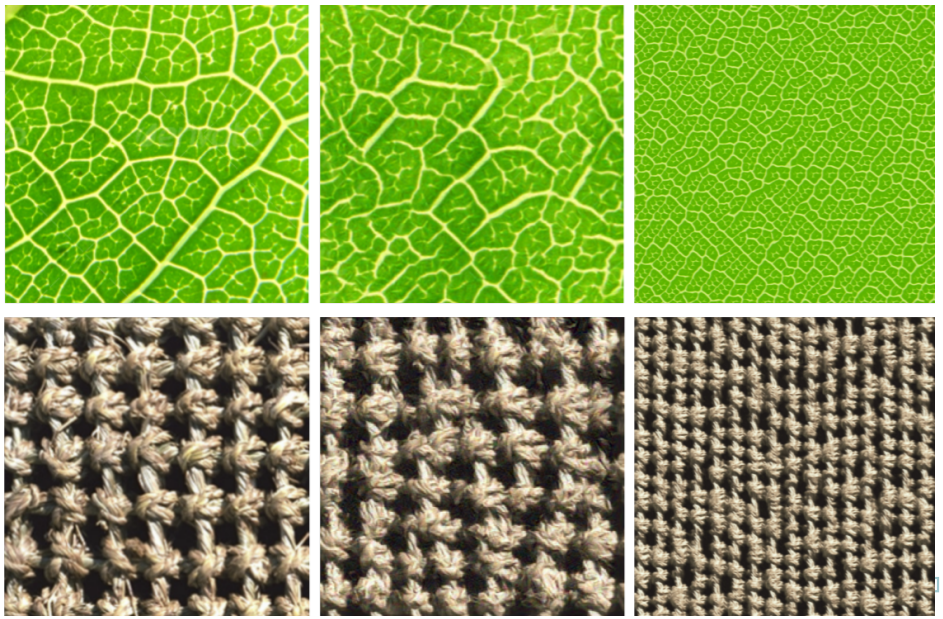
This test uses a more complex texture, with higher variability in the color distribution and thus less homogeneous. We can clearly see the poor results for the generative setting.



In order: Real one, VGG, Gauss, Gauss+VGG, Gauss+IV3, Gen-Gauss+VGG

All our experiments





Thanks

Merci pour votre attention!