TexTile: A Differentiable Metric for Texture Tileability

Supplementary Material

1. Introduction and Structure

In this supplementary material, we include extensive implementation details, additional results and visualizations.

The contents of this supplementary material can be divided as follows:

- In Section 2, we provide extensive implementation details of our model, training configuration, data augmentation, and inference.
- In Section 3, we show additional evaluations we performed to our model.
- In Figures 4 and 5, we show some images of our test set, in both tileable and non-tileable categories.
- In Figure 6, we show additional examples of our data augmentation policy.
- In Figures 7 and 8, we show full results of our comparisons with previous work on tileable texture synthesis, using the dataset in [15].
- In Figure 9, we show additional results of our tileable texture synthesis through outpainting method.
- In Figure 10, we show additional results of tileable texture generation through Neural Texture Synthesis.
- In Figure 11, we show additional results of tileable texture generation with single-image diffusion models.
- In Figures 12 to 17, we show additional results of our image alignment algorithm.
- In Figures 18 to 20, we show additional results of our repeating pattern detection algorithm.
- In Figures 21 and 22, we show model saliency maps for non-tileable and tileable textures, respectively.
- In Figures 23 to 34, we show filter visualizations of our model, on different layers. We use the *CNNLayerVisualization* class from the tool in the Convolutional Neural Network Visualizations Github Repository for this visualization, with additional modifications like gradient clipping or weight decay for improved results. These visualizations provide additional insights on the kinds of patterns the model is learning for detecting tileability.
- In Figure 35, we show results of an unsuccessful experiment of using circular padding within single-image diffusion models.

2. Implementation Details

2.1. Model Design

Our model builds upon a pretrained ConvNext [8]. In particular, we use the model architecture and weights `ConvNeXt Base Weights IMAGENET1K V1` from TorchVision [9]. To it, we introduce two residual and scaled multi-head Linear Self-Attention layers [17] to the 5th and 7th layers of the model. We use 16 heads in each layer, with 128 and 256 hidden units each, respectively. We selected linear self-attention due to their relatively slow computational cost, and placed them on the deeper layers of the convolutional backbone as this yielded the most cost-effective improvements. Placing them earlier in the model was computationally too expensive.

2.2. Training & Data Augmentation

As mentioned on the main paper, we train our models for 100 epochs using NAdam [5], Lookahead [18], Automatic Gradient Scaling and Mixed Precision Training [10], with an initial learning rate of 0.002, halved every 33 epochs. We use PyTorch [13] for training, and Kornia [14] for data augmentation. We use batch sizes of 24 samples, composed of balanced number of positive and negative examples. This process takes approximately 6 hours on a single Nvidia RTX 3060 GPU. We use images of $(384, 384)$ pixels for training and $(512, 512)$ for inference.

Hyperparameters for the data augmentation policy are tuned using Bayesian optimization [2]. For specific implementation details of each policy, please refer to Kornia [14]. We include the specific values below:

- **Color Augmentation**: We perform random color augmentation on HSV space. We randomly change the hue of the images (full hue range), their contrast (factor of 0.4), brightness (factor of 0.25), and saturation (factor of 0.25). We do this for every element in every batch ($p(\text{color augmentation}) = 1$).
- **Rescales**: We randomly rescale the input images, with different factors for vertical and horizontal dimensions, taken at random from the $(0.6, 1.4)$ range. We do this for every element in every batch ($p(\text{rescales}) = 1$).
- **Flips**: We randomly flip the images horizontally and/or vertically. We do this with probability: $p(\text{flip}_h) = p(\text{flip}_v) = 0.5$).
- **Blurs**: We randomly blur the input images with two different types of blurs: Gaussian and Box blurs. We set their kernel sizes to 7 and 3, respectively, and their probabilities to $p(\text{blur}_{\text{gaussian}}) = p(\text{blur}_{\text{box}}) = 0.15$.
- **Noise**: We apply random gaussian noise to the images with a probability of $p(\text{noise}) = 0.15$ and a $\sigma = 0.05$.
- **Equalization**: We randomly equalize the input images with a probability of $p(\text{equalize}) = 0.1$.
- **Random Gamma**: We randomly change the gamma the input images with a probability of $p(\text{gamma}) = 0.2$, with gamma and gain ranges selected at random in the range $(0.9, 1.1)$.
- **Elastic Transformations**: We randomly apply an elastic transformation of the non-tileable images with $p(elastic) = 0.15$, a kernel size of 15 and a $\sigma = 32$.
- **Affine Transformations**: We randomly apply affine transformations of the non-tileable images with $p(\text{affine}) = 0.25$, with rotations and shears randomly selected in the $(-20, 20)$ ranges.
- **Custom Augmentations**: For the policies explained in the main paper, we choose $p(\text{UnFold}) = 0.1$, $p(AUG_{F\to F'})$ and $p(AUG_{T\to F}) = 0.25$.

To ensure that each training batch is composed of the a balanced number of positive and negative examples given the previous probabilities, we select elements following:

$$p(T) = \frac{1 - 2p(AUG_{F\to F'})}{2p(AUG_{T\to F}) - 2p(AUG_{F\to F'})} \tag{2}$$

Where $p(T)$ is the probability of selecting an element from the training dataset composed of tileable examples.

As the last operation in our data augmentation and processing pipeline, we standardize each image with the ImageNet [4] standardization parameters.

2.3. Inference

As mentioned on the main paper, during inference, we apply a transformation to the logits of the model using a λ parameter. The logistic transformation has the goal of keeping TexTile on the $(0, 1)$ range, and λ helps*linearize* the TexTile values so they are more comparable and useful.

$$\text{TexTile} = \frac{1}{1 + \exp\left(-\lambda \cdot \mathcal{M}(\text{I}_{\text{tiled}})\right)} \tag{3}$$

We always set $\lambda = 0.25$ during inference. However, this may be changed at will. Its impact can be seen in Figure 1.

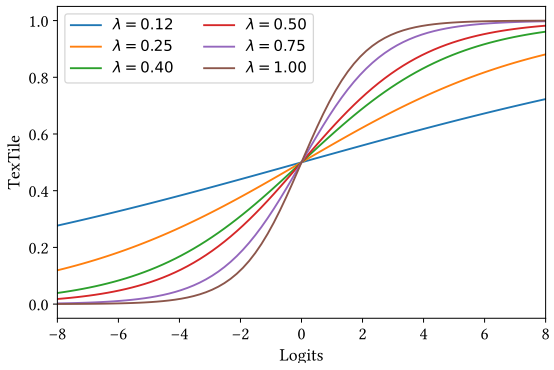

Figure 1. Impact of the value of λ to the value of TexTile.

3. Evaluation

Input Resolution

We aim to measure the influence of the input resolution in the performance of our models. To do so, we change the resolution of the tiled images I_{tiled} on the [64, 1024] range, and measure the classification performance on our test set, using ROC Area Under the Curve as our metric of choice. We show the results of this experiment in Figure 2. The model is generally robust to image resolution, showing very high generalization capabilities on images larger than 192 pixels. The maximum performance is achieved at medium size images, on the (384, 640) range. Tiny and very large resolutions somewhat hinder the performance of the model. This is probably due to the fact that very small resolutions may hide seamlessness problems in the texture borders, while larger images simply fall outside the range of resolutions that were used during training. However, even in extreme resolution scenarios, our model provides accurate estimations. We used (512, 512) resolution in all of the experiments in this paper and supplementary material.

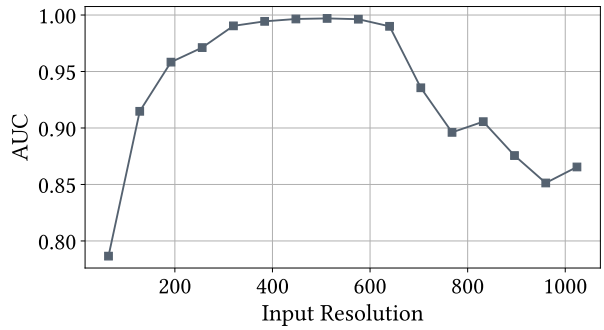

Figure 2. Impact of the resolution of the input images on the generalization of our model, measured on the test set.

Number of Repetitions

In this experiment, we aim to measure the number of repetitions (*i.e.* tilings) to apply to the input images I to generate I_{tiled}. Similarly to the previous experiment, we change the number of repetations in the tiled images I_{tiled} on the [1, 10] range, and measure the classification performance on our test set, using ROC Area Under the Curve. We show the results of this experiment in Figure 2, where evaluation was done using (512,512) pixel images. The model is generally robust to this parameter, as the number of repetitions shown during training is selected as random, making the model invariant to this factor. We observe that even if the images are not repeated (repetitions=1), the model performs better than random, which suggests that the model is able to leverage factors other than repeating artifacts for its estimations, like illumination or alignment problems. Further, we see an optimal performance at 2-6 repetitions. With more repetitions than 6, our model predictions start to degrade, albeit they are still accurate. We can thus conclude that the model does not need to see more than 2 repetitions to obtain accurate estimations. We use 2x2 tilings in every experiment in the paper and supplementary material.

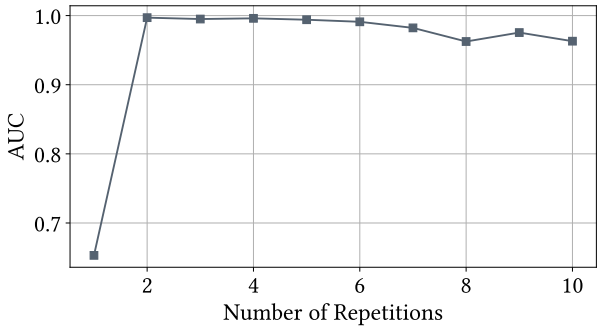

Figure 3. Impact of the number of repetitions applied to the input images on the generalization of our model, measured on the test set.

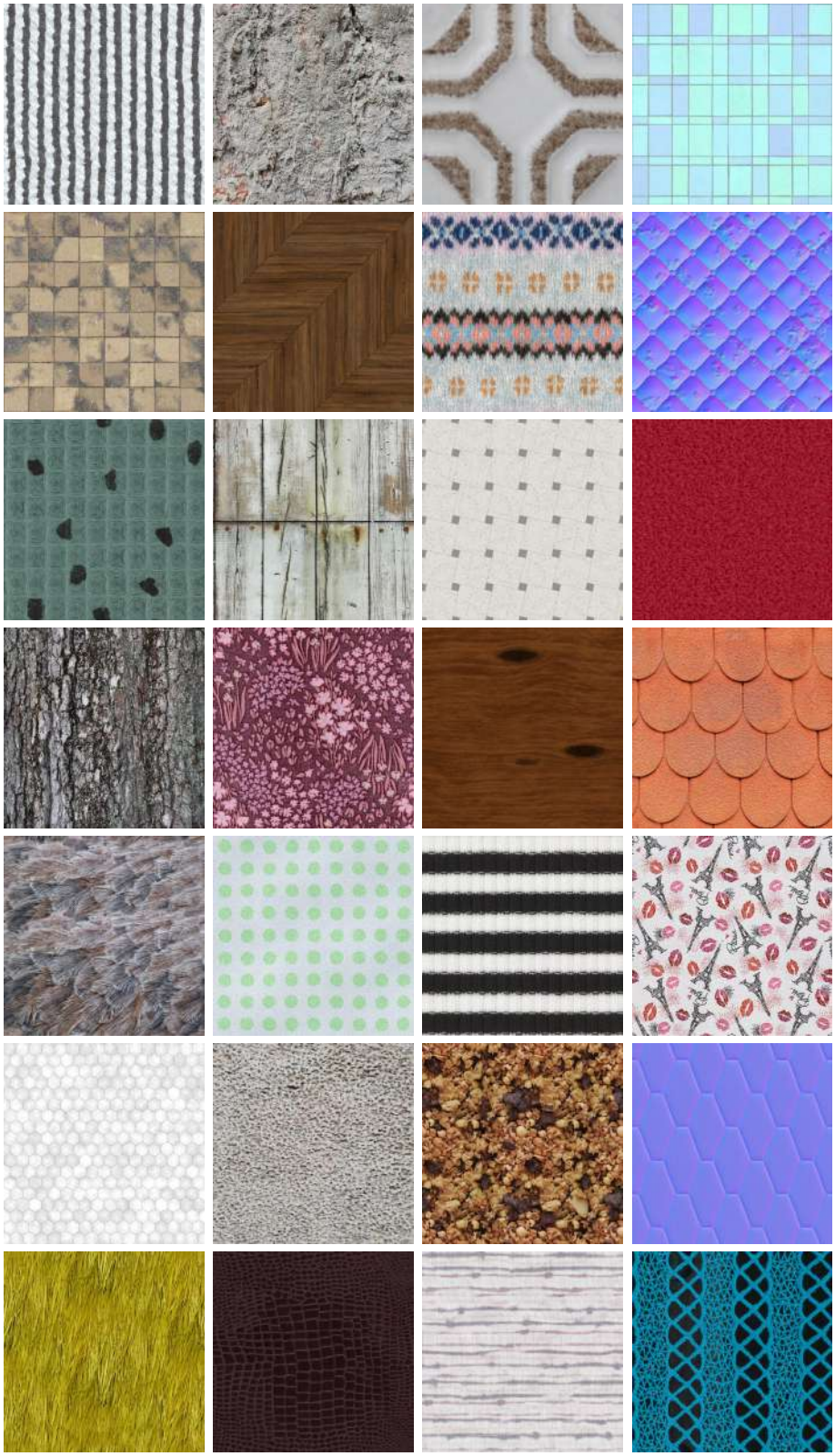

Figure 4. Some images of our test set, with the category of tileable textures.

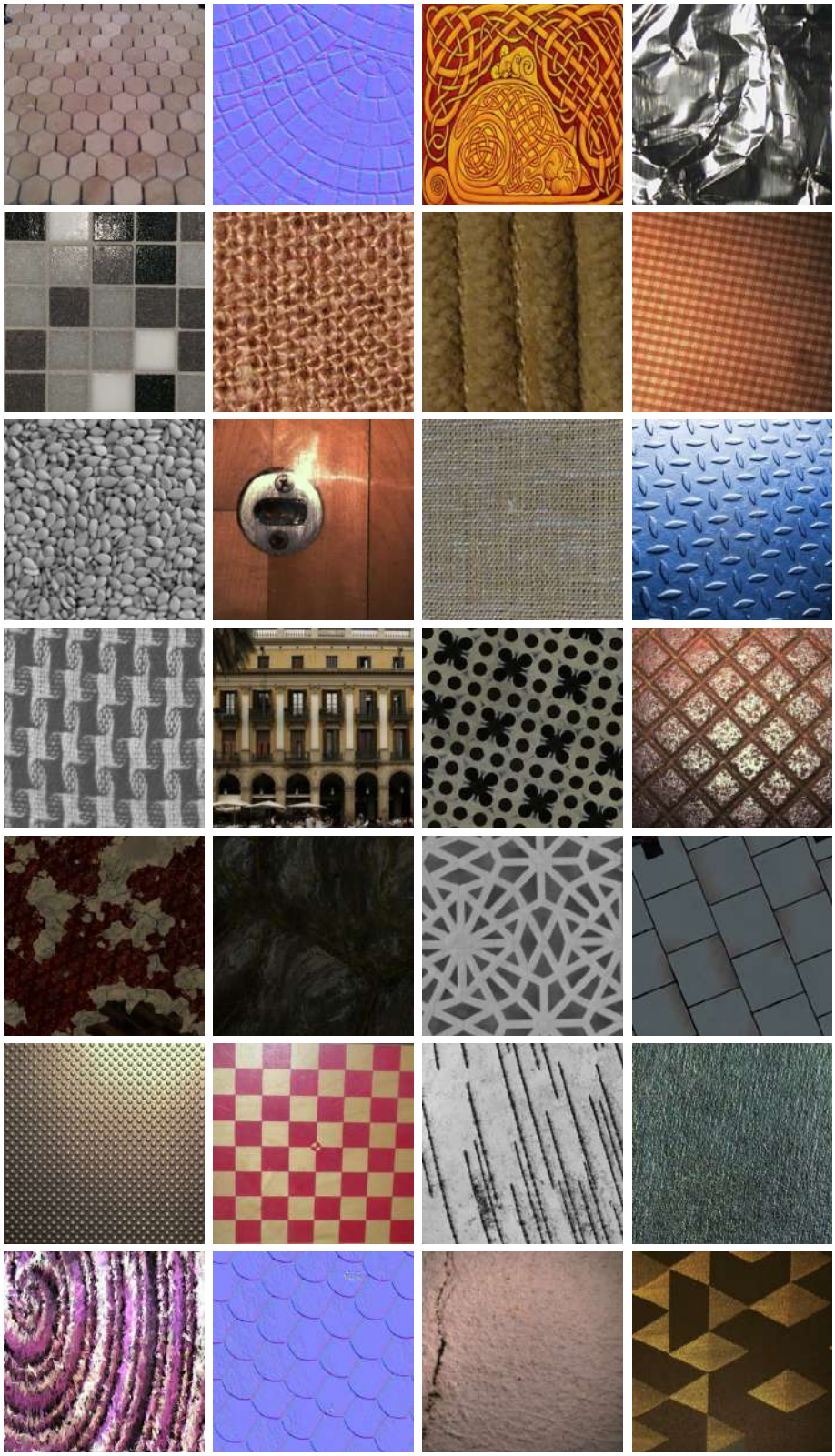

Figure 5. Some images of our test set, with the category of non-tileable textures.

Figure 6. Additional examples of our data augmentation policy. From a tileable texture (top), our data augmentation can generate tileable (middle) and non-tileable (bottom) variations.

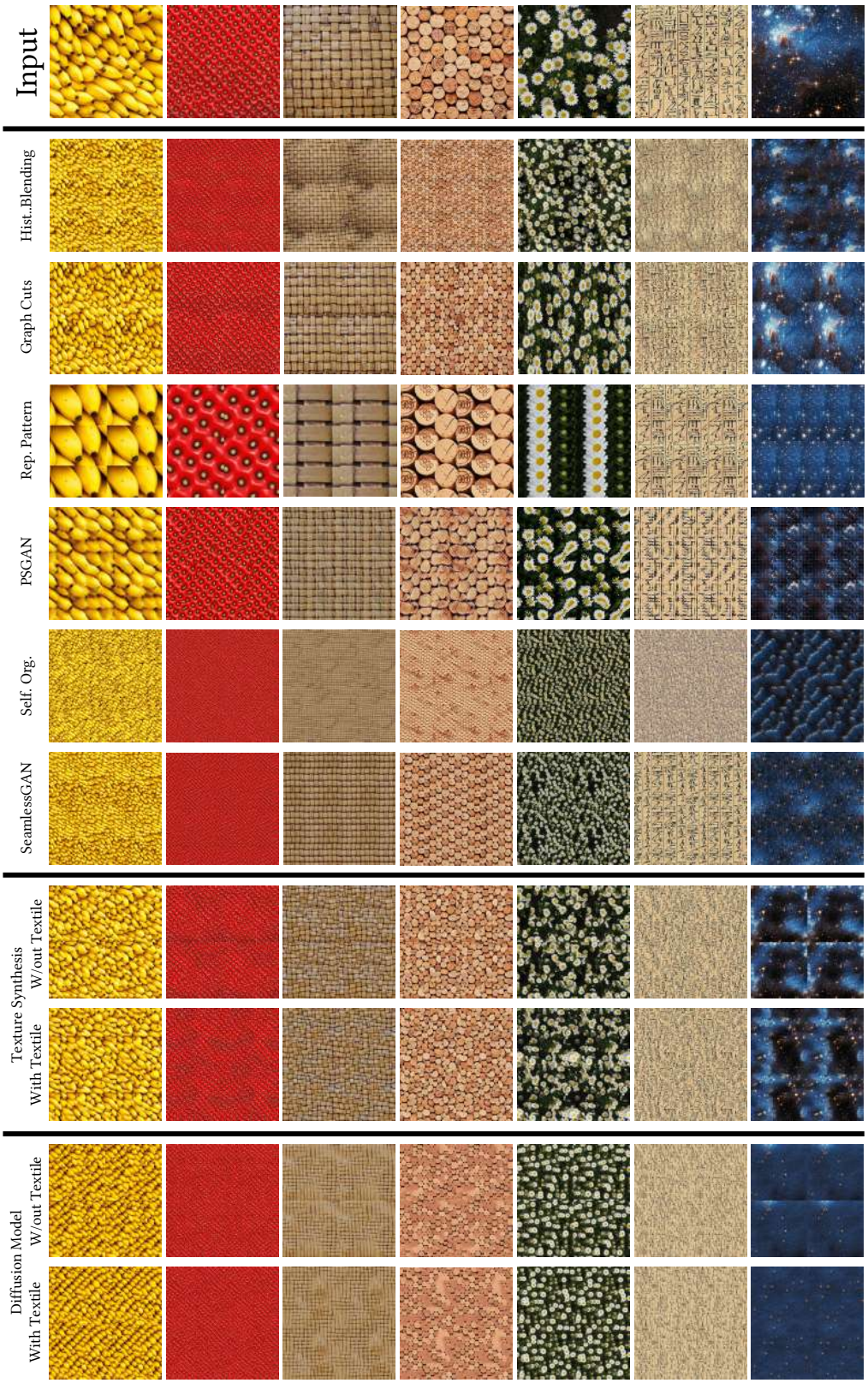

Figure 7. From top to bottom: Input images, results by Histogram Blending [3], Graph Cuts [7], Repeating Pattern Detection [16], PS-GAN [1] , Self-Organising Textures [12], SeamlessGAN [15], Neural Texture Synthesis [6] without and with Textile, and SinFusion [11] without and with Textile.

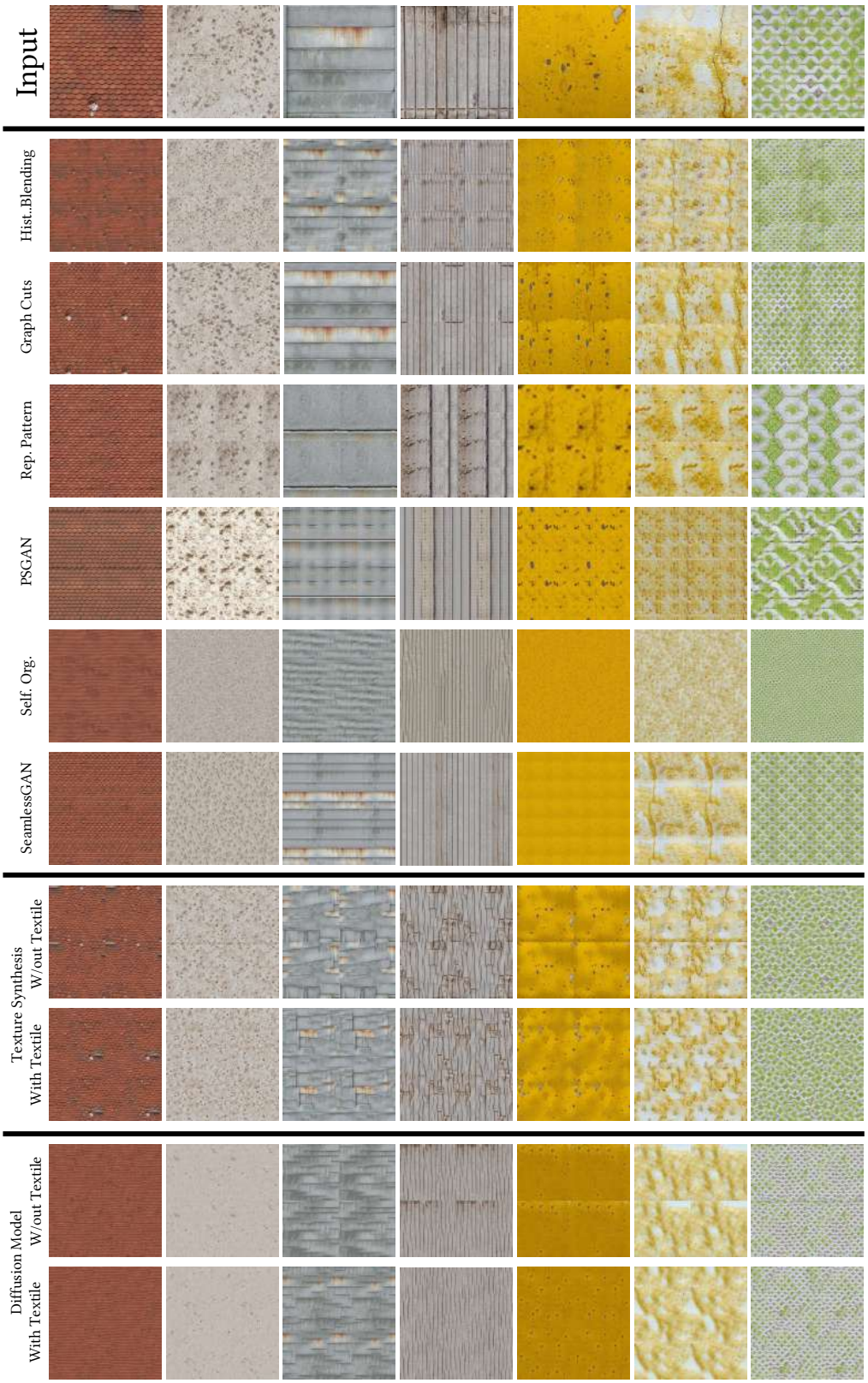

Figure 8. From top to bottom: Input images, results by Histogram Blending [3], Graph Cuts [7], Repeating Pattern Detection [16], PS-GAN [1] , Self-Organising Textures [12], SeamlessGAN [15], Neural Texture Synthesis [6] without and with Textile, and SinFusion [11] without and with Textile.

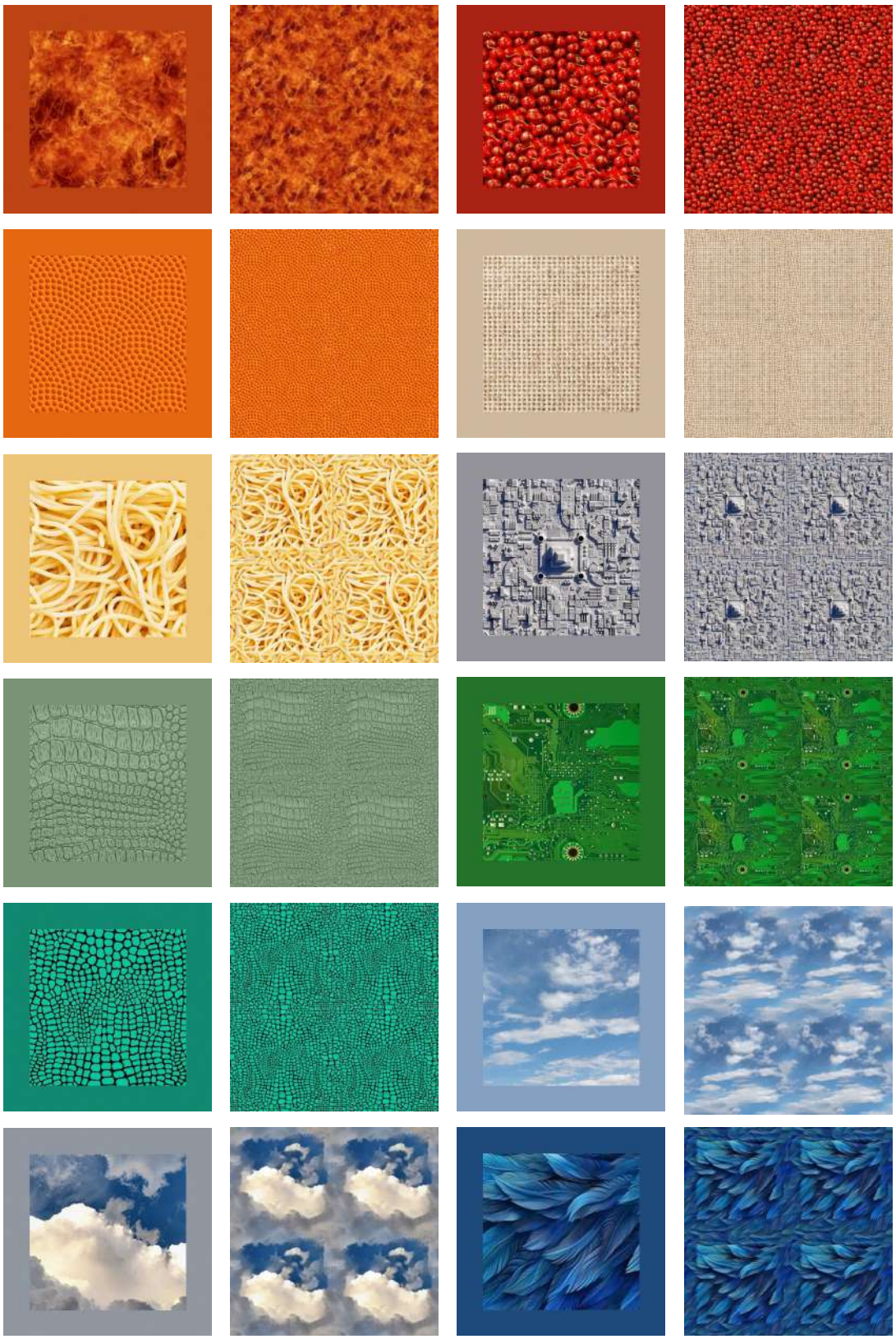

Figure 9. Additional results of our method for tileable texture synthesis using outpainting, leveraging TexTile as a loss for tileability and Neural Texture Synthesis [6] as the generative process. On the left, *non-tileable* input images with the area to be outpainted in a solid color; on their sides, outpainted results, obtained by maximizing tileability, shown in a 2x2 tile composition.

Figure 10. Additional comparisons of Neural Texture Synthesis with and without TexTile as a loss function. For each column, on the left, we show input images, on the middle, the baseline result (tiled 2x2) of [6], on the right, the result (tiled 2x2) of our modification of [6] which uses TexTile as an additional loss function.

Figure 11. Additional comparisons of a Diffusion Model evaluated with and without TexTile. For each column, on the left, we show input images, on the middle, the baseline result (tiled 2x2) of [11], on the right, the result (tiled 2x2) of our modification of [11] which uses TexTile as a loss function during the diffusion process.

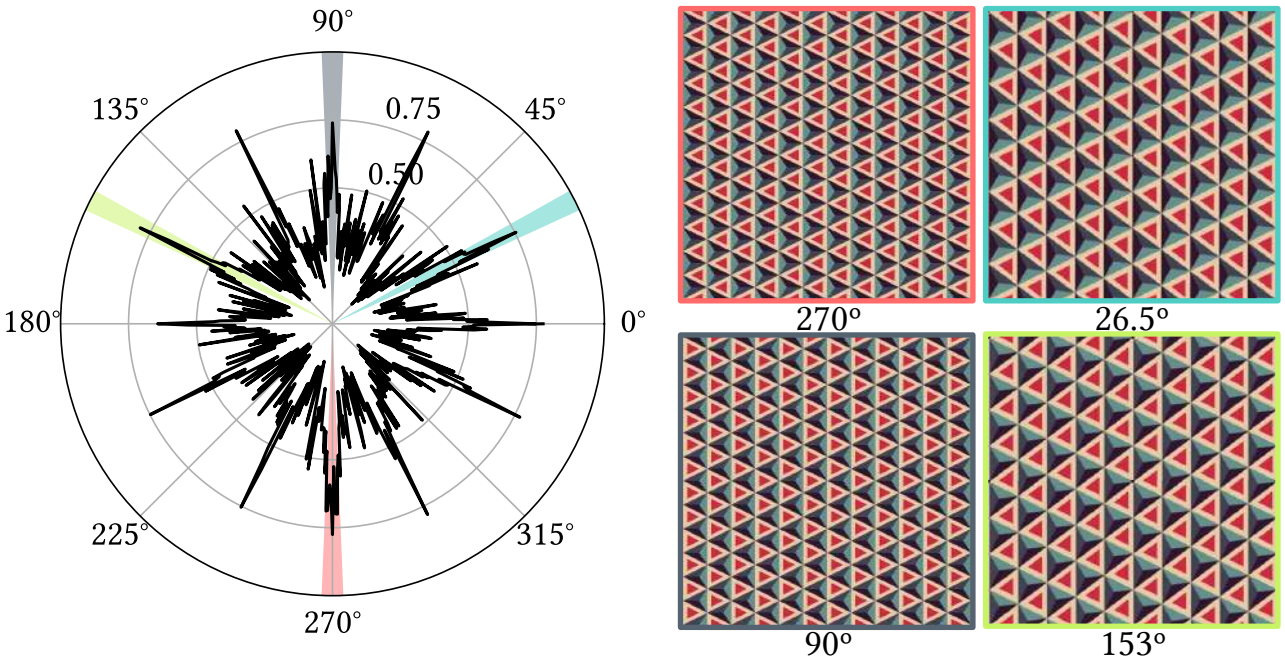

Figure 12. On the left, TexTile under different rotation angles. On the right, samples of rotated images on different local peaks (tiled 2x2).

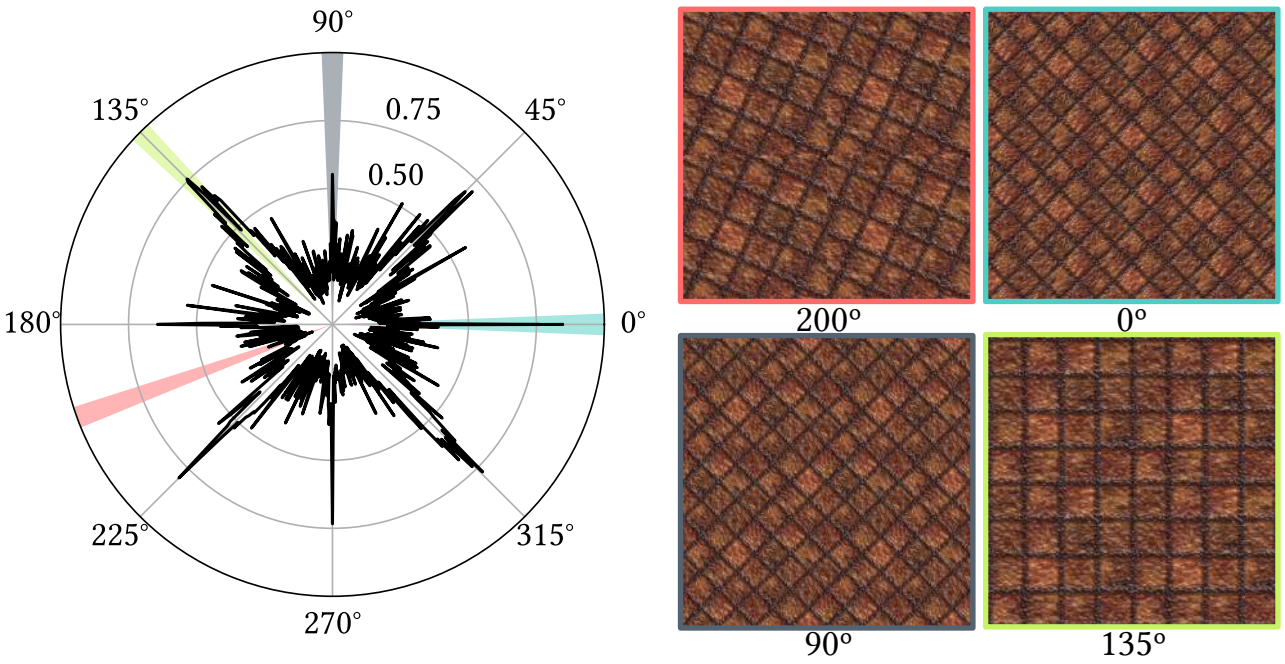

Figure 13. On the left, TexTile under different rotation angles. On the right, samples of rotated images on different local peaks (tiled 2x2).

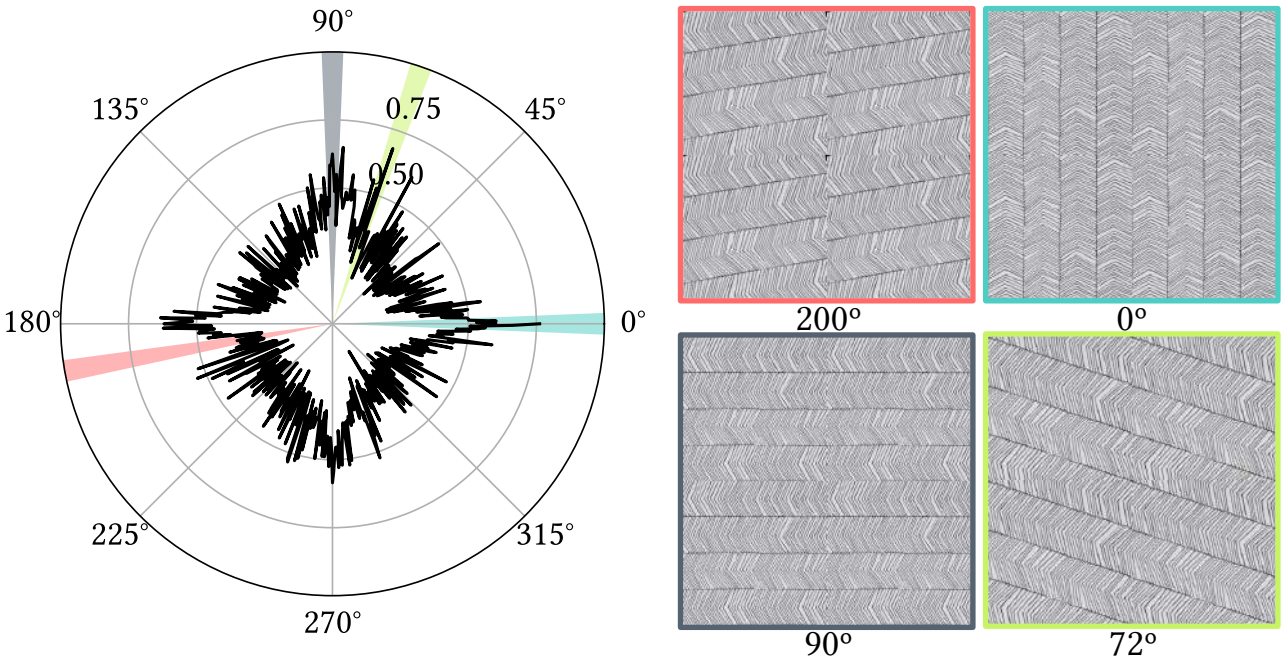

Figure 14. On the left, TexTile under different rotation angles. On the right, samples of rotated images on different local peaks (tiled 2x2).

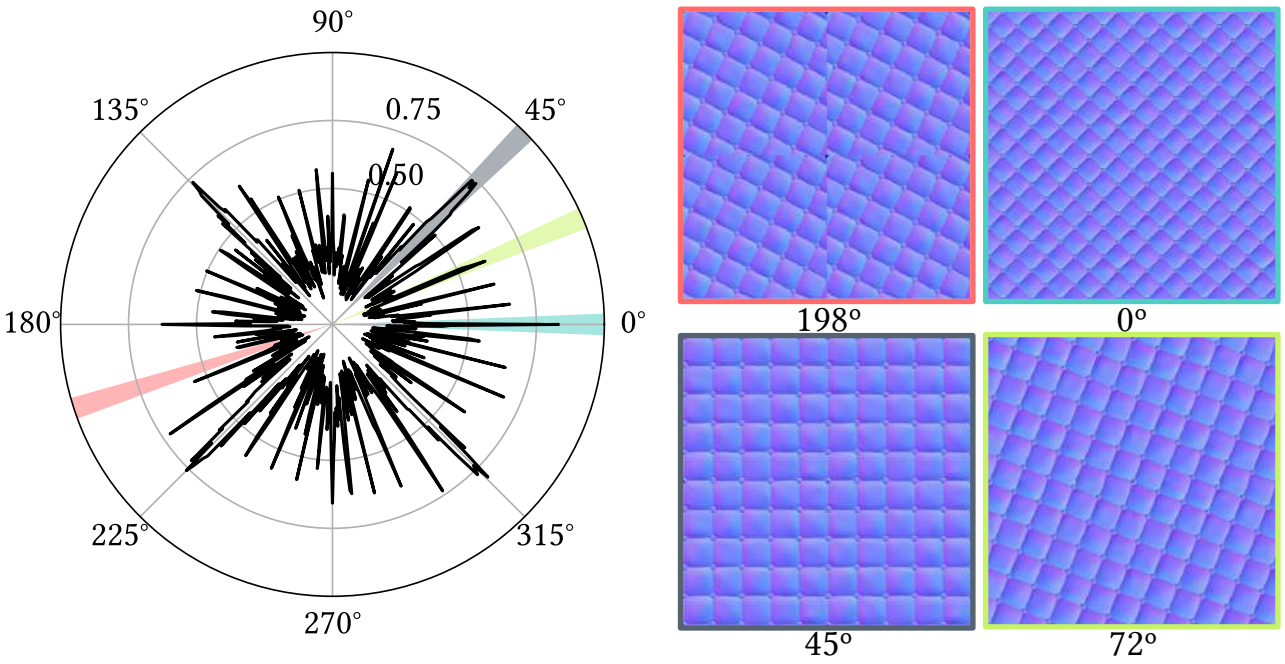

Figure 15. On the left, TexTile under different rotation angles. On the right, samples of rotated images on different local peaks (tiled 2x2).

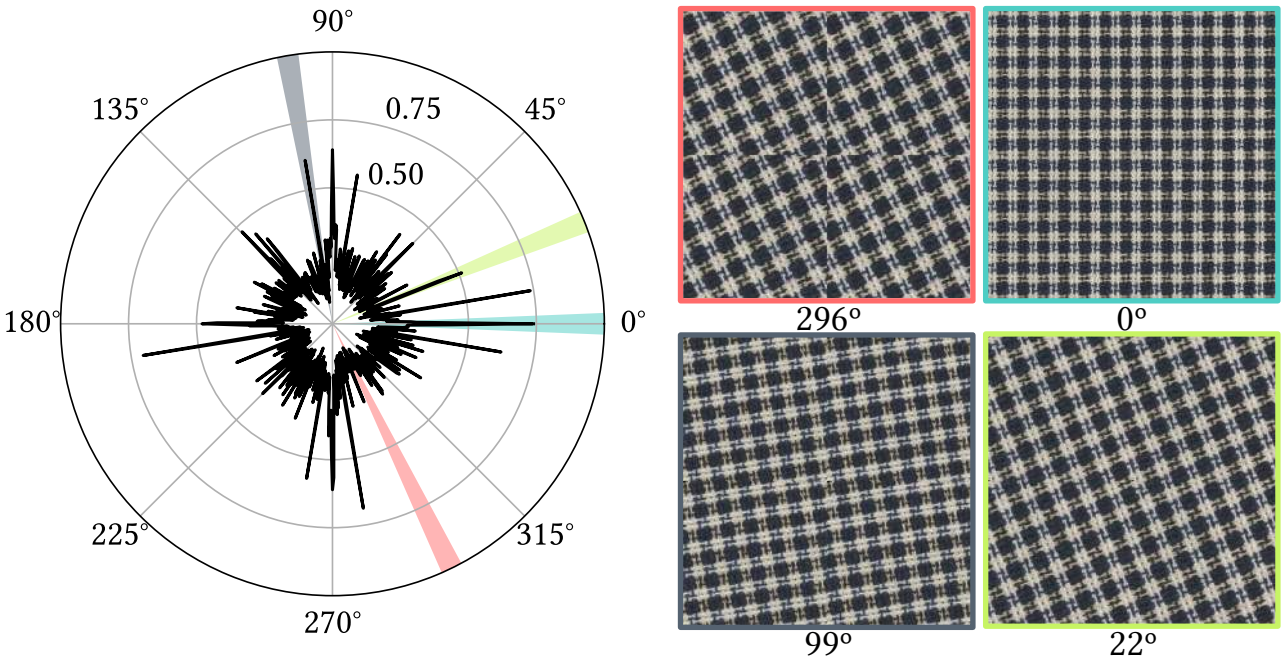

Figure 16. On the left, TexTile under different rotation angles. On the right, samples of rotated images on different local peaks (tiled 2x2).

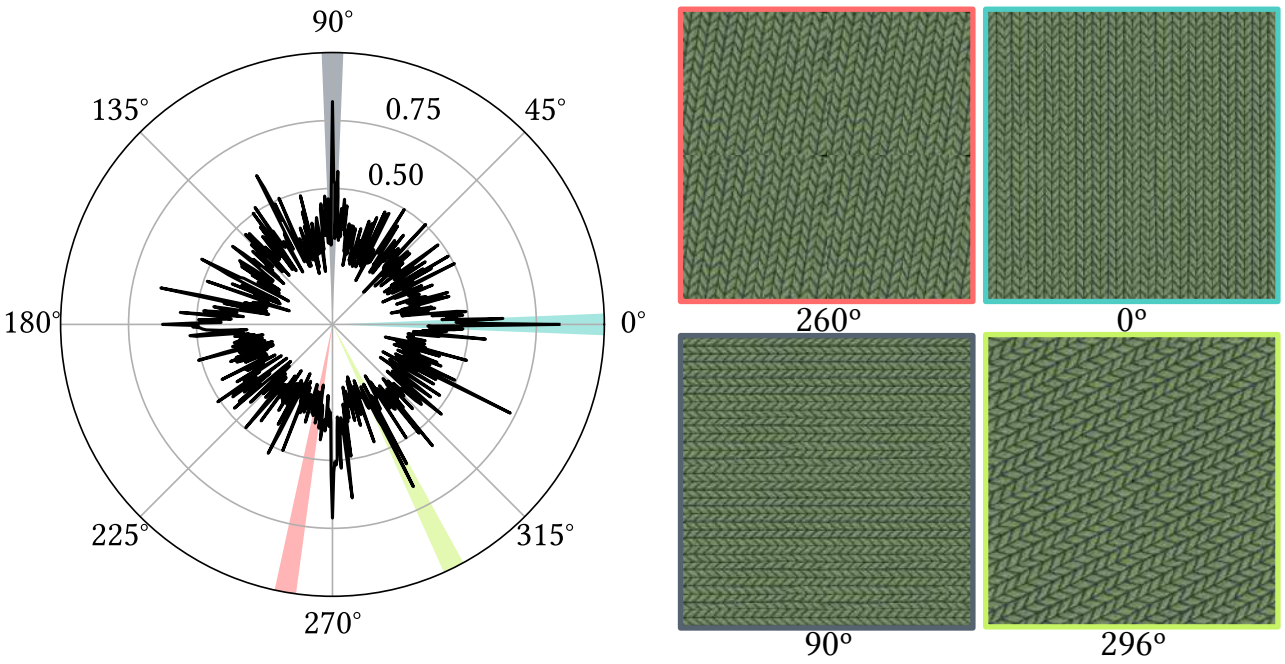

Figure 17. On the left, TexTile under different rotation angles. On the right, samples of rotated images on different local peaks (tiled 2x2).

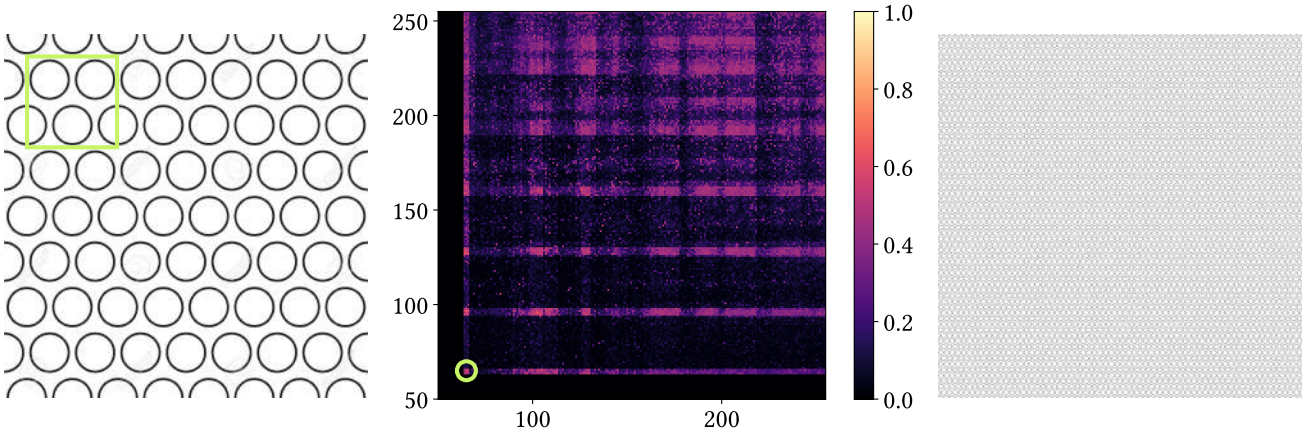

Figure 18. On the left, input image. In the middle, TexTile values for different crop sizes. The optimal crop is highlighted on both images with a green inset. On the right, the optimal crop, tiled many times for visualization.

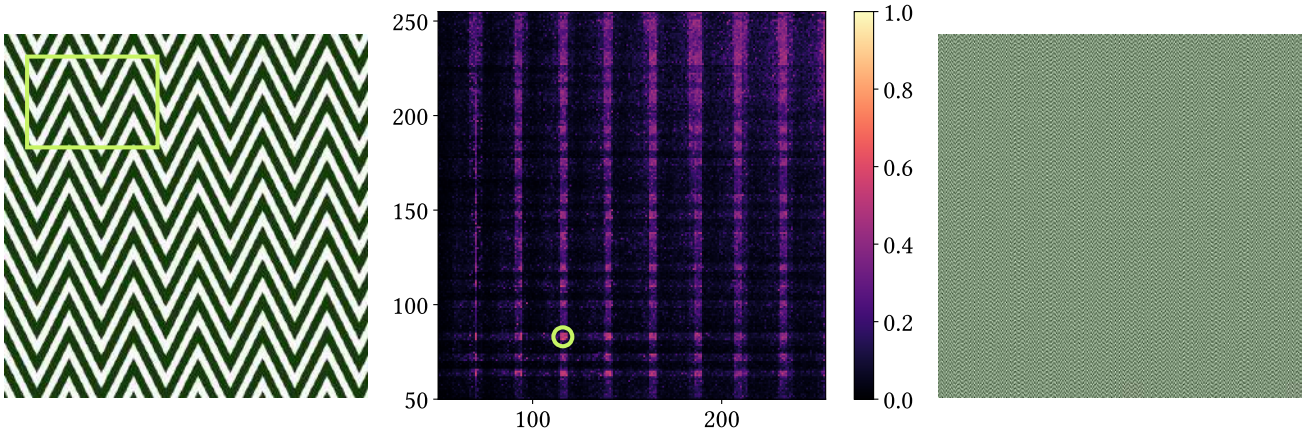

Figure 19. On the left, input image. In the middle, TexTile values for different crop sizes. The optimal crop is highlighted on both images with a green inset. On the right, the optimal crop, tiled many times for visualization.

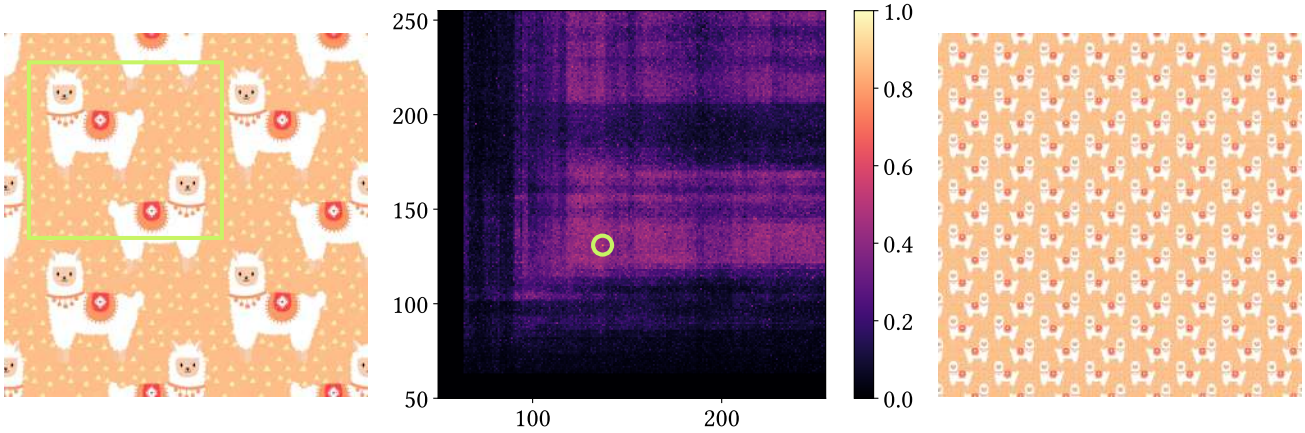

Figure 20. On the left, input image. In the middle, TexTile values for different crop sizes. The optimal crop is highlighted on both images with a green inset. On the right, the optimal crop, tiled many times for visualization.

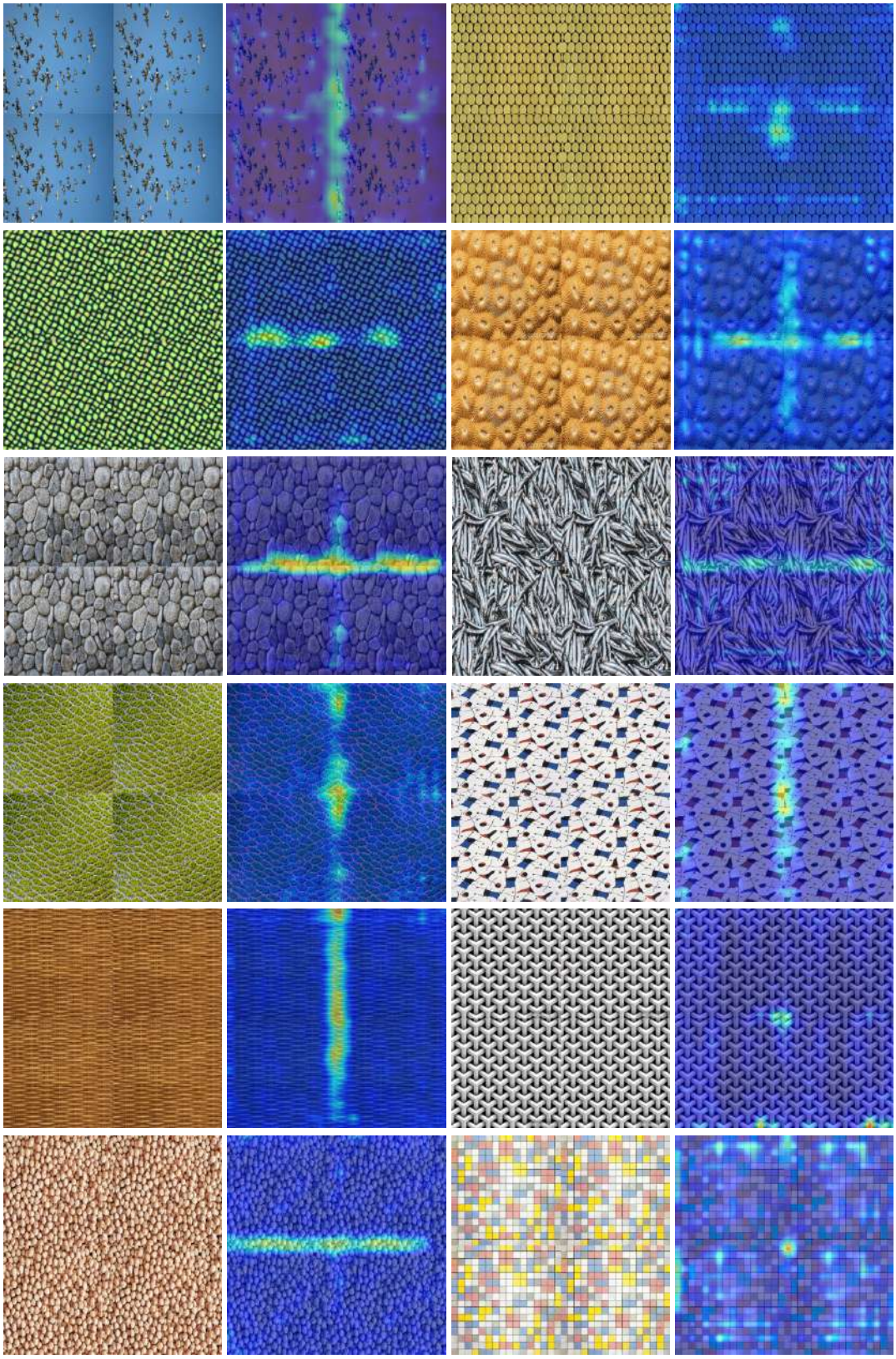

Figure 21. Saliency maps of our model for different textures which are not seamlessly tileable.

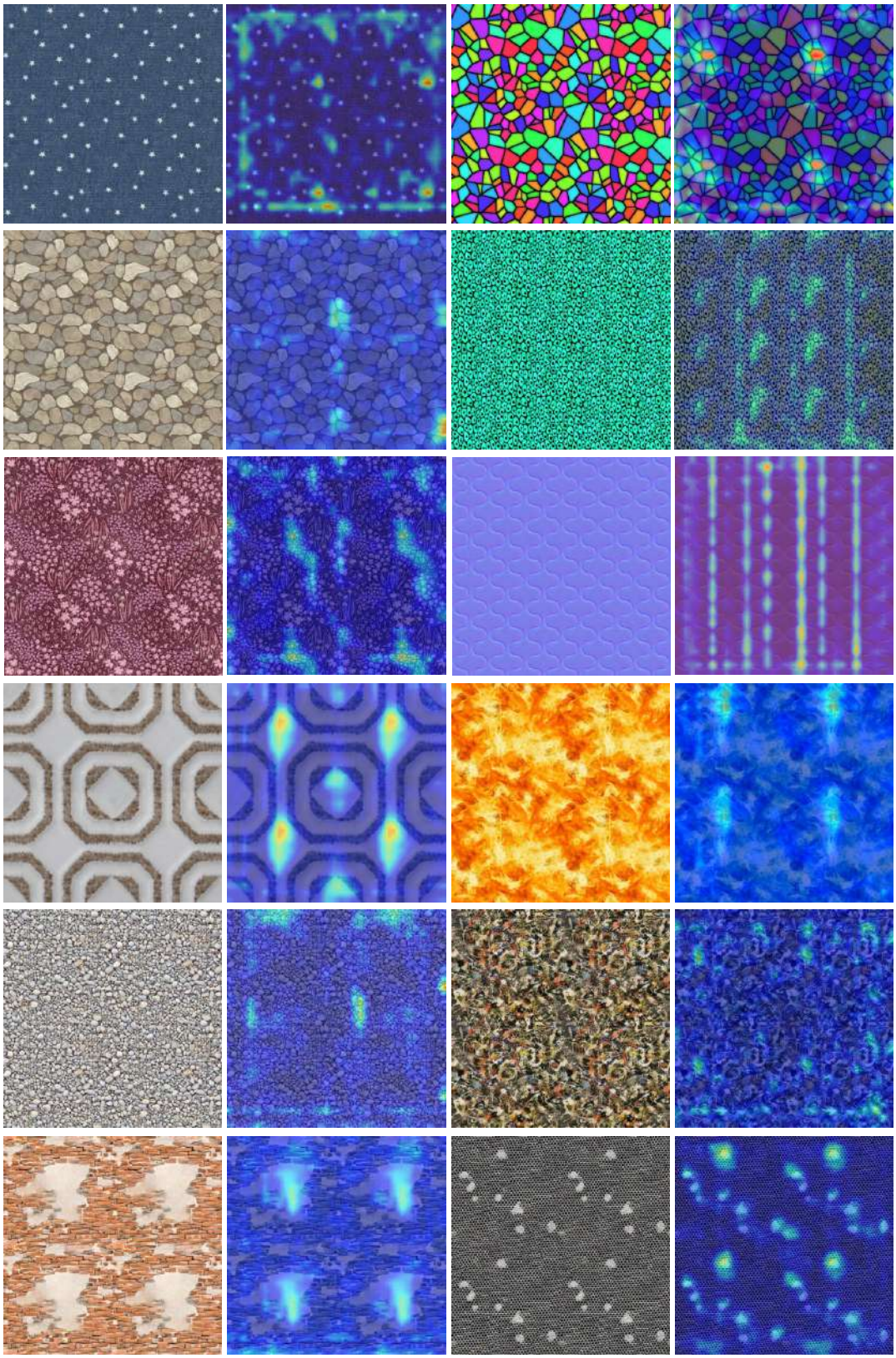

Figure 22. Saliency maps of our model for different textures which are seamlessly tileable. For these cases, the model is leveraging repeating artifacts in the images that may hinder their tileability.

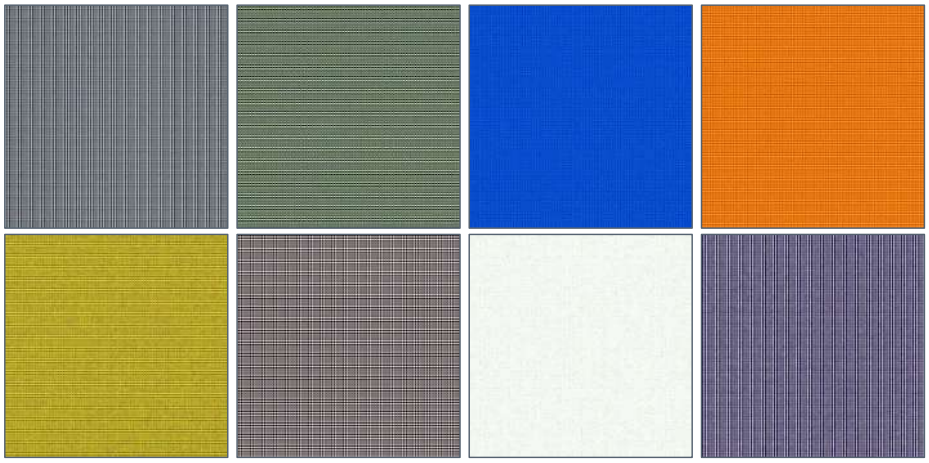

Figure 23. Visualization of some filters in the first layer of our model. Most of the filters in the first layer seem very similar, measuring local intensities, colors, and very local edges.

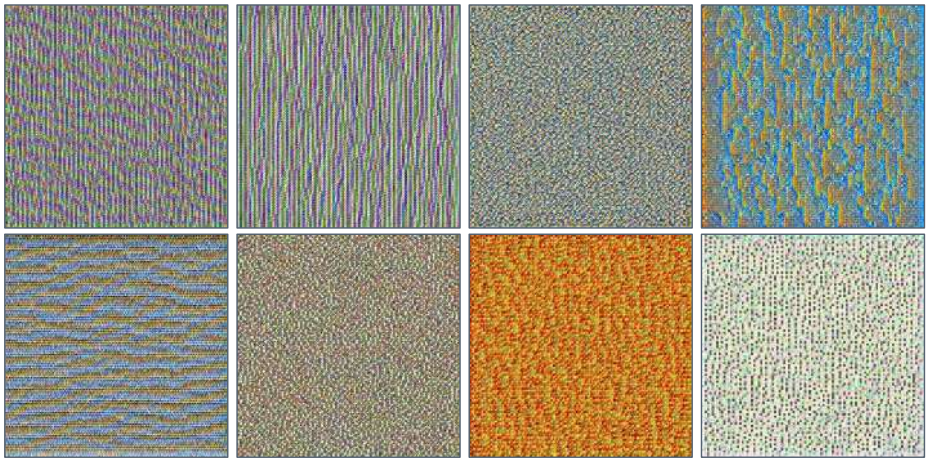

Figure 24. Visualization of some filters in the second layer of our model. These filters appear to measure edges and local textures.

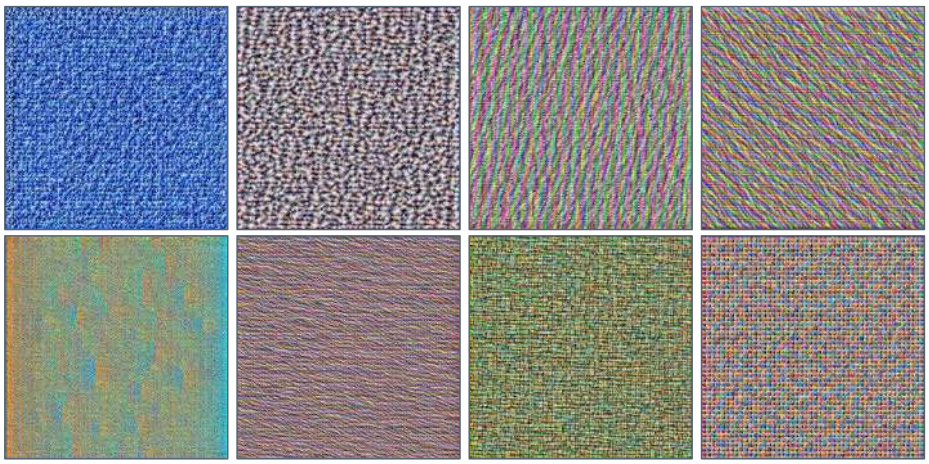

Figure 25. Visualization of some filters in the third layer of our model. As in the second layer, these filters appear to measure edges and local textures.

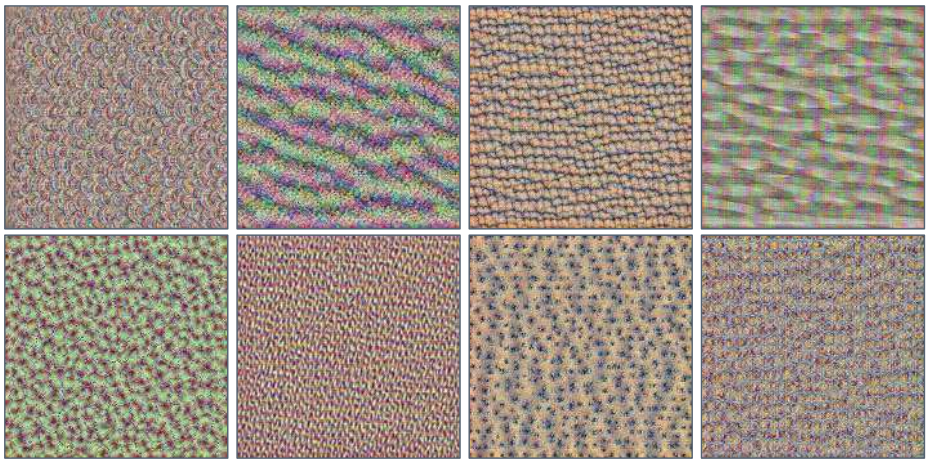

Figure 26. Visualization of some filters in the fourth layer of our model. Notice how some interesting textures start to emerge.

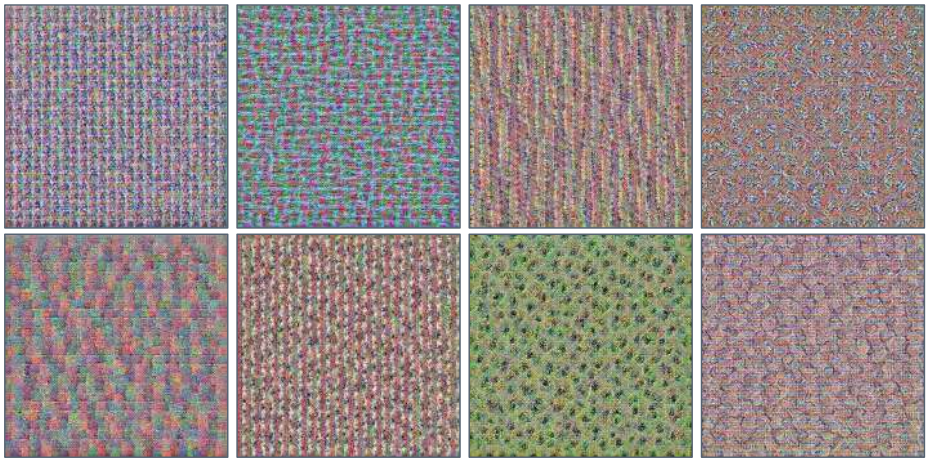

Figure 27. Visualization of some filters in the fifth layer of our model. Notice how some interesting grid patterns start to emerge.

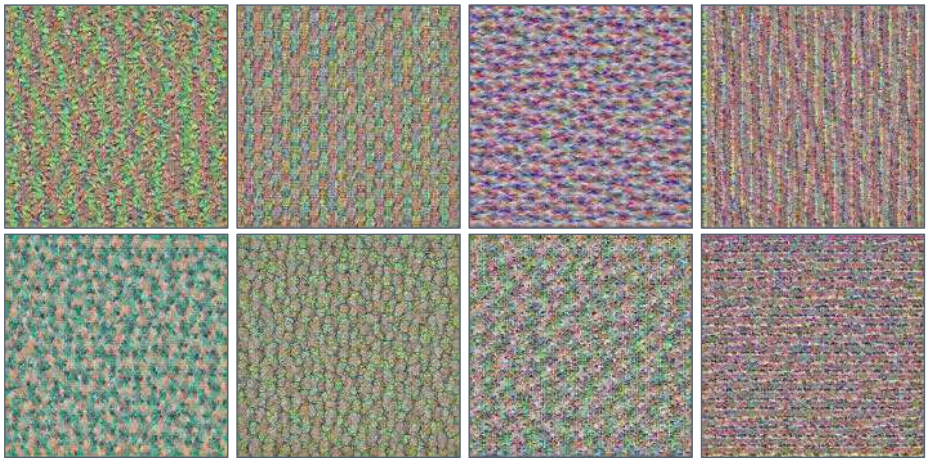

Figure 28. Visualization of some filters in the sixth layer of our model. Notice how some texture patterns emerge.

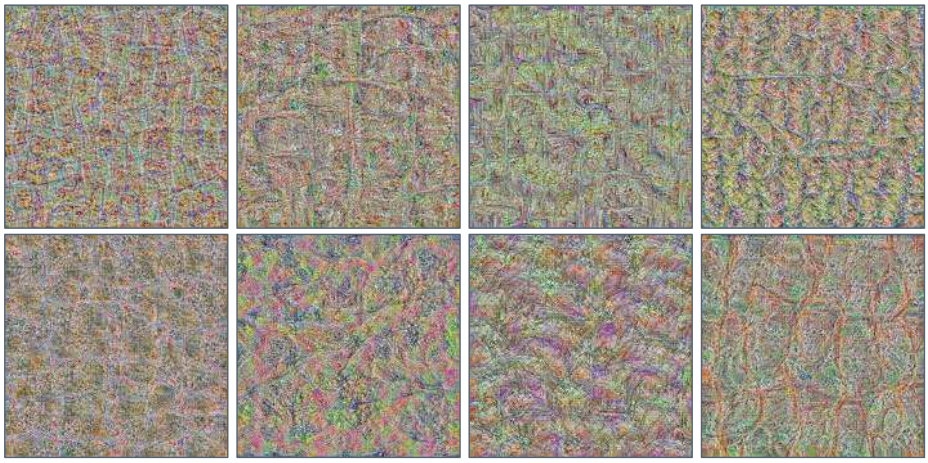

Figure 29. Visualization of some filters in the seventh layer of our model. This is the first self-attention layer in our model. In comparison with previous layers, the patterns in these filters are more global, spanning much larger areas in the images.

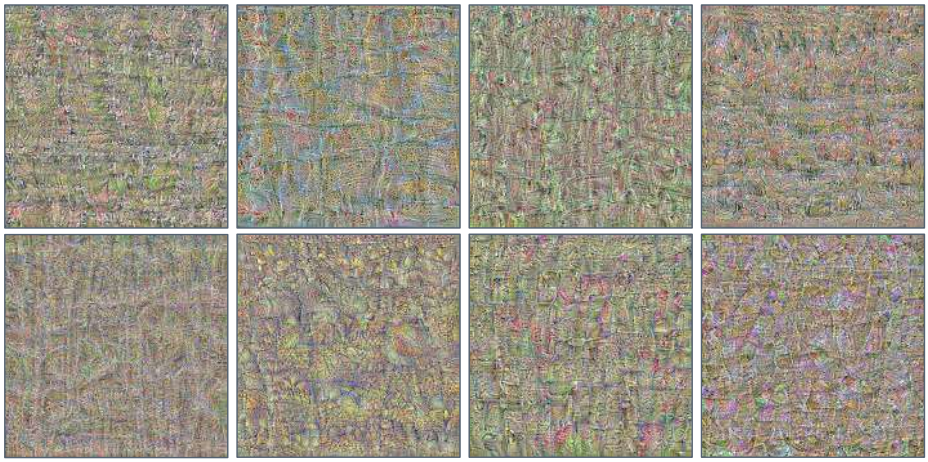

Figure 30. Visualization of some filters in the eighth layer of our model. Similarly to the seventh layer, the patterns in these filters are global, spanning large areas in the images. From this point onward, the patterns start to lose any clear semantic meaning.

Figure 31. Visualization of some filters in the ninth layer of our model.

Figure 32. Visualization of some filters in the tenth layer of our model.

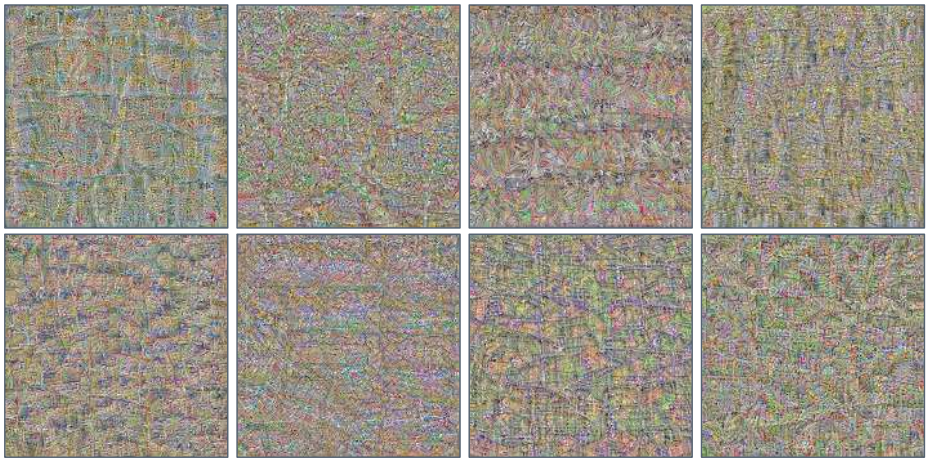

Figure 33. Visualization of some filters in the eleventh layer of our model. This is the second and last self-attention layer in our model.

Figure 34. Visualization of some filters in the twelfth layer of our model.

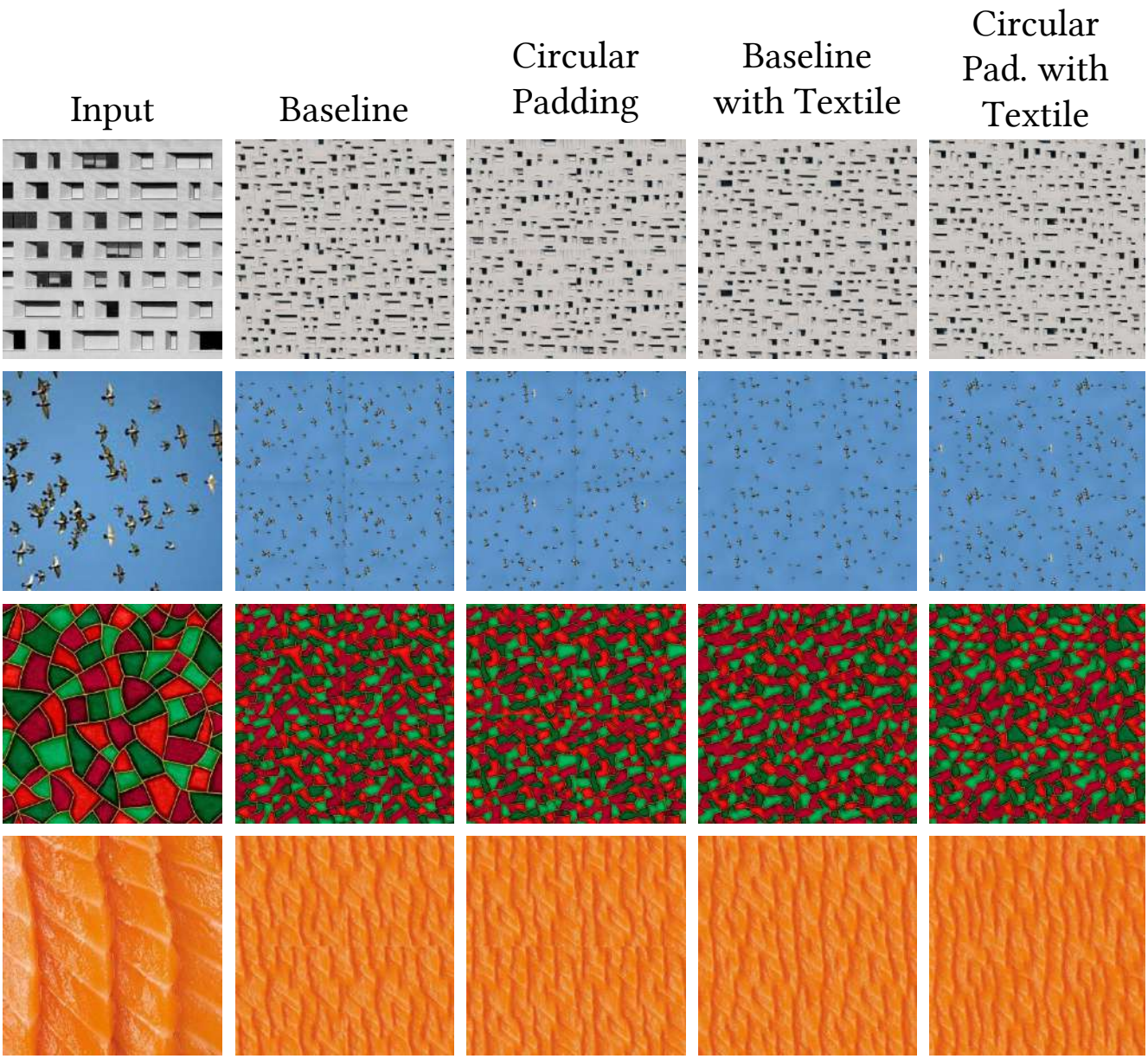

Figure 35. An experiment on circular padding. We changed the baseline implementation of SinFusion [11], changing the architecture to use circular padding in the convolutional layers, inspired by this idea for Stable Diffusion. We did not achieve tileable textures with this simple modification unless we introduced TexTile as a loss function during inference.

References

[1] Urs Bergmann, Nikolay Jetchev, and Roland Vollgraf. Learning texture manifolds with the periodic spatial gan. pages 469–477, 2017. 7, 8

[2] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com. 2

[3] Thomas Deliot and Eric Heitz. Procedural stochastic textures by tiling and blending. *GPU Zen*, 2, 2019. 7, 8

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 248–255. Ieee, 2009. 2

[5] Timothy Dozat. Incorporating nesterov momentum into adam. 2016. 2

[6] Eric Heitz, Kenneth Vanhoey, Thomas Chambon, and Laurent Belcour. A sliced wasserstein loss for neural texture synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9412–9420, 2021. 7, 8, 9, 10

[7] Zhengqin Li, Mohammad Shafiei, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and svbrdf from a single image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2475–2484, 2020. 7, 8

[8] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11976–11986, 2022. 2

[9] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1485–1488, 2010. 2

[10] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. In *International Conference on Learning Representations*, 2018. 2

[11] Yaniv Nikankin, Niv Haim, and Michal Irani. Sinfusion: Training diffusion models on a single image or video. In *International Conference on Machine Learning*. PMLR, 2023. 7, 8, 11, 22

[12] Eyvind Niklasson, Alexander Mordvintsev, Ettore Randazzo, and Michael Levin. Self-organising textures. *Distill*, 2021. https://distill.pub/selforg/2021/textures. 7, 8

[13] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 2

[14] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3674–3683, 2020. 2

[15] Carlos Rodriguez-Pardo and Elena Garces. Seamlessgan: Self-supervised synthesis of tileable texture maps. *IEEE Transactions on Visualization and Computer Graphics*, 2022. 1, 7, 8

[16] Carlos Rodriguez-Pardo, Sergio Suja, David Pascual, Jorge Lopez-Moreno, and Elena Garces. Automatic extraction and synthesis of regular repeatable patterns. *Computers & Graphics*, 83:33–41, 2019. 7, 8

[17] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020. 2

[18] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. *Advances in neural information processing systems*, 32, 2019. 2