

Scattering Transform as invertible function

Uri Ben-Hur, Binyamin Mayan

Department of Mathematics
Technion institution

Abstract. In this paper we present a modified model for the Scattering Transform, that is invertible. We point some properties of the Scattering Transform preserved in the new model. Additionally, we will show some empiric results supporting those claims and present our results of running classification and texture reconstruction tasks. We modified an exiting Scattering Transform python library called Kymatio to support the invertible model.

1. Introduction

The Scattering Transform is a specific realization of a convolutional network, where the filters are chosen as wavelets. The Scattering Transform has some very useful properties that makes it a powerful transformation in pattern recognition.

Scattering Transform definition (Levie, 2024)¹:

1. Denote the dilation – rotation space, by scales smaller than $J \in \mathbb{Z}$, by

$$\Lambda_J := \mathbb{Z}_{>J} \times K$$

2. Given $m \in \mathbb{N}_0$, we call a sequence $p = (\lambda_i, \dots, \lambda_m) \in \Lambda_J^m$ a path. We call

$$P_J = \bigcup_{m \in \mathbb{N}_0} \Lambda_J^m$$

the space of all paths. Note that \emptyset is also a path, as it belongs to Λ_J^0 . We define

$$P_J^m = \bigcup_{j=0}^{m-1} \Lambda_J^j.$$

3. For $\lambda \in \Lambda_J$, and $f \in L^2(\mathbb{R}^2)$, define $U_J(\lambda)f = |(W_J f)(\cdot, \lambda)| = |f * \eta_\lambda|$.

4. The scattering operator U_J is defined to be the operator that maps signals $f \in L^2(\mathbb{R}^2)$ to functions $P_J \rightarrow \mathbb{R}$, as follows.

For each $p = (\lambda_1, \dots, \lambda_m) \in P_J$

$$\begin{aligned}(U_J f)(p) &= U_J(p)f = U_J(\lambda_m) \cdots U_J(\lambda_2)U_J(\lambda_1)f \\ &= |\cdots ||f * \eta_{\lambda_1}| * \eta_{\lambda_2}| * \cdots * \eta_{\lambda_m}|.\end{aligned}$$

Define

$$U_J(\emptyset)f = f.$$

5. The windowed scattering transform S_J is defined to be the operator that maps signals $f \in L^2(\mathbb{R}^2)$ to functions $P_J \rightarrow \mathbb{R}$, as follows. For each $p = (\lambda_1, \dots, \lambda_m) \in P_J$

$$(S_J f)(p) = S_J(p)f = U_J(p)f * \xi_J.$$

Note that

$$S_J(\emptyset)f = f * \xi_J.$$

6. Denote by $S_J(P_J)f$ the sequence

$$S_J(P_J)f := \{(S_J f)(p)\}_{p \in P_J}.$$

Following that definition, we can see that the windowed Scattering Transform takes general signals, and outputs a sequence of low-band signals.

2. Invertible Scattering Transforms

Because of the use of norm in the classic Scattering Transform it is not invertible.

To make the model invertible, we use the following approach:

In each layer of the transform, instead of using the norm operator, which is not invertible, we use a different operator, that is both invertible and gives out a positive signal.

The idea is to split the signal into 4 new signals: first by taking the Real and Imaginary parts of the signal, and then applying the Relu operator on each one of the resulting signals and their negation.

Formally, given a signal f we define a new operator

$$G(f) = \left(\text{Relu}(\text{Re}(f)), \text{Relu}(-\text{Re}(f)), \text{Relu}(\text{Im}(f)), \text{Relu}(-\text{Im}(f)) \right)$$

$$\text{when } \text{Relu}[f](x) = \begin{cases} f(x) & \text{if } f(x) \geq 0 \\ 0 & \text{if } f(x) < 0 \end{cases}$$

We also define how to apply G to multiple signals: $G(\{f_i(x)\}_{i=1}^n) := \left\{ G(f_i(x)) \right\}_{i=1, j=1}^{n,4}$

we will show the definition of the invertible Scattering Transform, only rewriting the sections changed from the original definition and using the same notations.

Definition of invertible Scattering Transform:

3. For $\lambda \in \Lambda_j, k \in [4]$, and $f \in L^2(\mathbb{R}^2)$ define:

$$\widetilde{U}_j(\lambda, k)f = G\left((W_j f)(\cdot, \lambda)\right)_k = G(f * \eta_\lambda)_k$$

4. The scattering operator \widetilde{U}_j is defined to be the operator that maps signals $f \in L^2(\mathbb{R}^2)$ to functions $P_j \rightarrow \mathbb{R}$, as follows:

For each $p = (\lambda_1, \dots, \lambda_m) \in P_j$, $\tilde{k} = (k_1, \dots, k_m) \in [4]^m$

$$\begin{aligned} (\widetilde{U}_j f)(p) &= \widetilde{U}_j(p, \tilde{k})f = \widetilde{U}_j(\lambda_m, k_m) \cdots \widetilde{U}_j(\lambda_2, k_2) \widetilde{U}_j(\lambda_1, k_1)f \\ &= G_{k_m}(\dots G_{k_2}(G_{k_1}(f * \eta_{\lambda_1}) * \eta_{\lambda_2}) * \dots * \eta_{\lambda_m}). \end{aligned}$$

Define

$$\widetilde{U}_j(\emptyset, k)f = f.$$

5. The windowed scattering transform \tilde{S}_j is defined to be the operator that maps signals $f \in L^2(\mathbb{R}^2)$ to functions $P_j \rightarrow \mathbb{R}$, as follows. For each $p = (\lambda_1, \dots, \lambda_m) \in P_j$,

$\tilde{k} = (k_1, \dots, k_m) \in [4]^m$

$$(\tilde{S}_j f)(p, \tilde{k}) = \tilde{S}_j(p, \tilde{k})f = \widetilde{U}_j(p, \tilde{k})f * \xi_j.$$

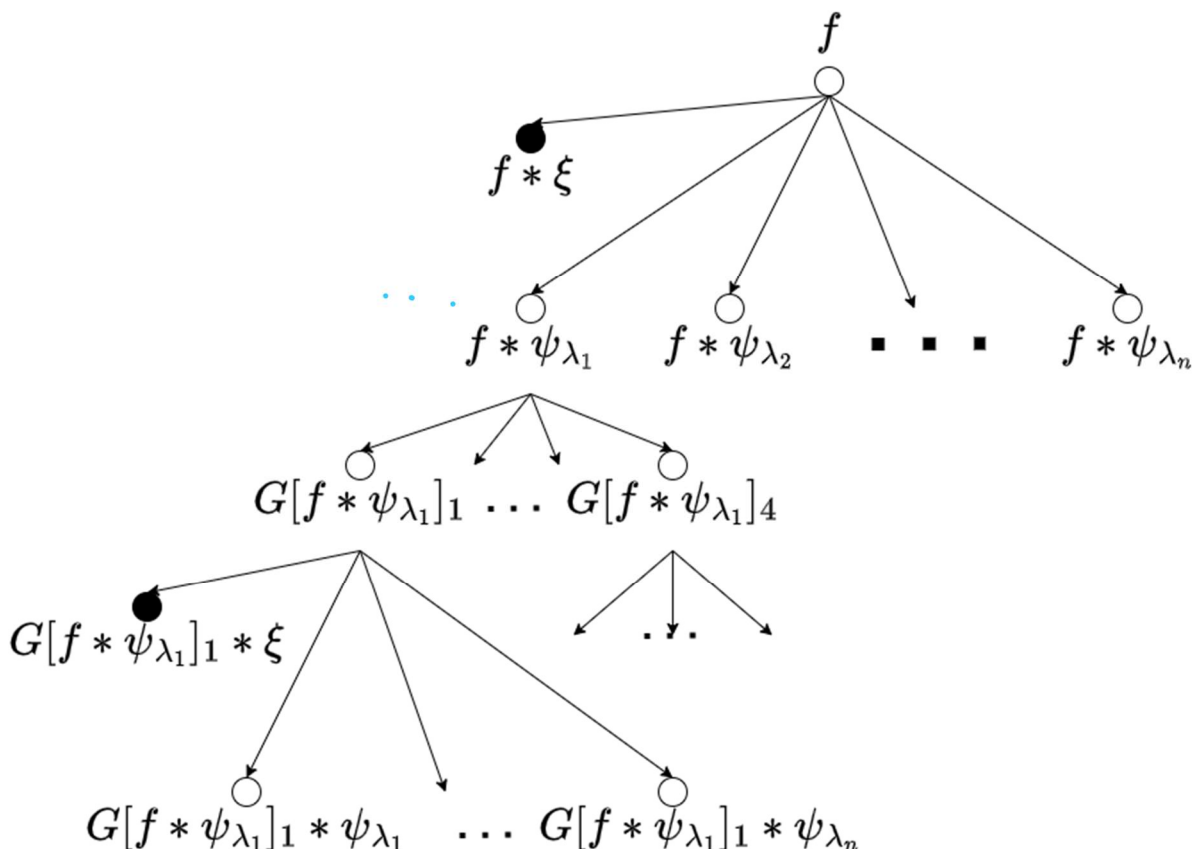
Note that

$$\tilde{S}_j(\emptyset, k)f = f * \xi_j.$$

6. Denote by $\tilde{S}_j(P_j)f$ the sequence

$$\tilde{S}_j(P_j)f := \{(\tilde{S}_j f)(p, \tilde{k})\}_{p \in P_j, \tilde{k} \in [4]^N}.$$

We can look at the Invertible Scattering Transform in a visual way, by drawing it as a rooted tree, where each black node is an "end node" meaning one of the outputs of the transform, and white nodes are "inner nodes" representing some stages along the computation of the transform.



3. Kymatio library

The Kymatio library² is an implementation of the wavelet Scattering Transform in the Python programming language, suitable for large-scale numerical experiments in signal processing and machine learning.

Kymatio uses Morlet wavelets as filters, and has several backend implementations, using different python packages.

To implement the Invertible Scattering Transform we forked the Kymatio library and modified it to support the new model. We only added this functionality to the NumPy and PyTorch backends.

Additionally, the Kymatio library supports calculating the Scattering Transform only to a depth of 2, that is only for paths of length 2 in P_f^2 . In our implementation we enabled calculation of the Invertible Scattering Transform up to any desired depth, we did so by using recursion instead of the original iterative way.

Andreux M., Angles T., Exarchakis G., Leonarduzzi R., Rochette G., Thiry L., Zarka J., Mallat S., Andén J., Belilovsky E., Bruna J.,² Lostanlen V., Chaudhary M., Hirn M. J., Oyallon E., Zhang S., Cella C., Eickenberg M. (2020). Kymatio: Scattering Transforms in Python. *Journal of Machine Learning Research* 21(60): 1–6, 2020

4. Properties of Scattering Transforms

We know that the Scattering Transform has the following properties:

- Mapping of information to low frequencies
- Non-expensiveness

We know theoretically that the above properties should also hold for the Invertible Scattering Transform.

In this section we will show some empiric results we got, supporting these theoretical claims.

Non expansiveness

short explanation

First let's start with the definition of non-expensiveness.

definiton: a mapping $F: X \rightarrow Y$ between two metric spaces X and Y is called non – expensive if $\forall x, y \in X$:

$$dist_Y(F(x), F(y)) \leq dist_X(x, y)$$

Now as proven in Levie's lecture notes³ the Scattering Transform is non expensive as an operator from $L^2(\mathbb{R}^2)$ with the L^2 norm, to $L^2(\mathbb{R}^2)^{P_J}$ with the following norm

$$\|h\|_2 := \sqrt{\sum_{p \in P_J} \|h_p\|_2^2} \quad \text{where } h := \{h_p\}_{p \in P_J}$$

We also expect the Invertible Scattering Transform to be non-expensive.

The empiric process

We wrote a python script⁴ that does as follow:

- Choosing n random MNIST images.
- Applying the Invertible Scattering Transform on each image.
- For any 2 images, we calculated:
 - The distance between the pictures= $dist_{image}$ (thinking about them as vectors in \mathbb{R}^m and taking the Euclidean distance)
 - The distance between the coefficients= $dist_{coef}$ (again by looking at them as vectors)
 - We then added the expansion factor $\frac{dist_{image}}{dist_{coef}}$ into a list
- Finally, we calculated the min and max and average values of the expansion factors over all pairs of images

Levie, Ron. "Convolutional Networks on Grids and Graphs: Mathematical Analysis from a Signal Processing Perspective". 2024. Math. ³
Theory of Convolutional Neural, Technion
kymatio_main/examples/2d/check_non_expensiveness.py ⁴

As we saw, to be non-expensive the distance between the coefficients should be smaller or equal than the distance between the images, hence we expect $\frac{dist_{image}}{dist_{coef}} \geq 1$.

So, to make sure none of the picture pairs 'break's' the non-expensiveness property we want to check that the minimum value of $\frac{dist_{image}}{dist_{coef}}$ is bigger or equal to 1.

The results

After running the script on a 100 random images from MNIST, we got a min contraction factor of 12.79 , a max contraction factor of 62.86 and an average contraction factor of 24.09.

We can see, that for this limited experiment we got a significantly big minimal expansion factor, supporting the claim that the Invertible Scattering Transform is non-expensive.

Mapping to low frequencies

short explanation

If we look at grayscale images as signals $f: L^2(\mathbb{R}^2) \rightarrow \mathbb{R}$ with a finite support, we can then take its Fourier Transform giving us the contribution of each frequency in the image. Where lower frequencies are closer to the origin.

So, what we expect to see in the Invertible Scattering Transform, is that as we look at dipper layers of the transform, and apply on their resulting coefficient the Fourier transform, we get signals that are more and more concentrated around 0.

The empiric process

We wrote a python script ⁵that does as follow:

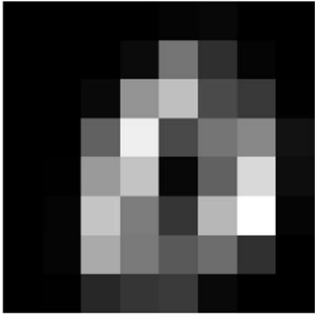
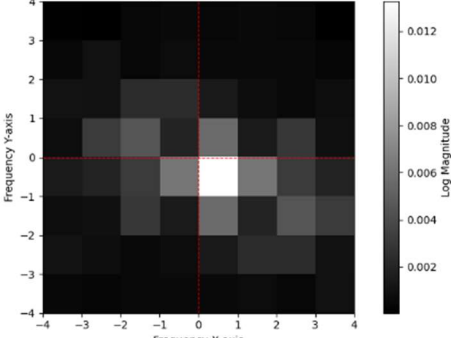
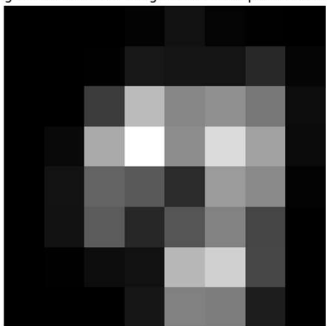
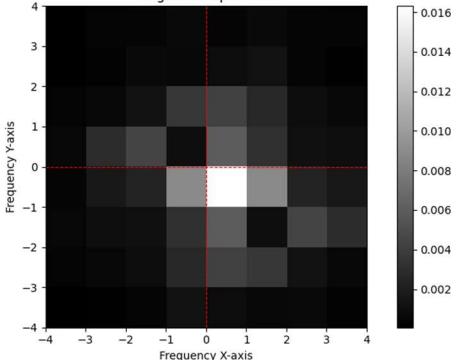
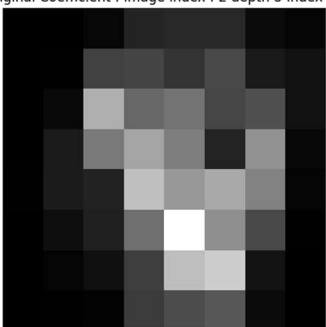
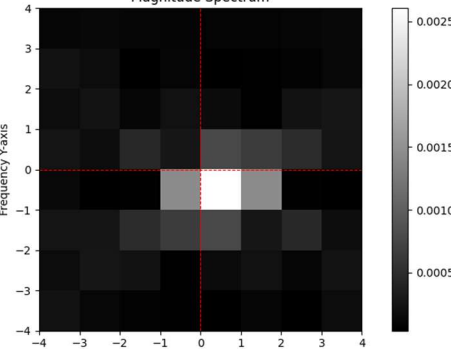
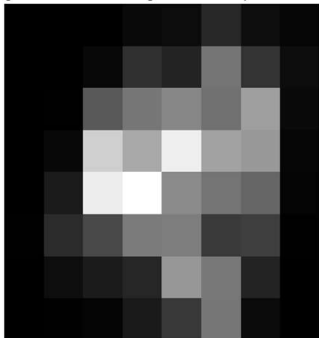
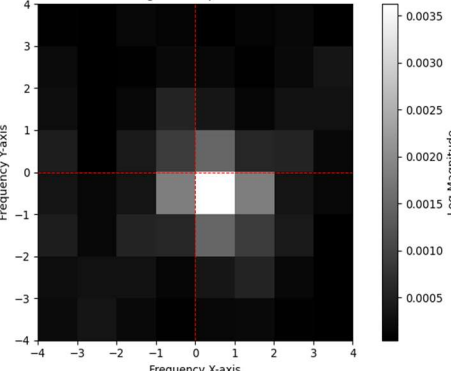
- Takes a random MNIST image
- Compute the Invertible Scattering Transform on the image, with a depth of 3
- Take coefficients from different levels in the tree
- Compute the Fourier transform on the coefficients
- Normalize the output and visualize the coefficient and its Fourier transform

The results

The figures below show 4 different coefficients and their Fourier Transform (in the left and right respectively), in depth 2 and 3 of the Invertible Scattering Transform.

More samples from the above experiment, can be found in the appendix.

We can see that indeed most of the signal is concentrated around 0, and that as we go dipper in the tree, we get lower and lower frequencies.

depth	Frequencies comparison	
2	<p>Original Coefficient : image index : 1 depth 2 index : 1</p> 	<p>Magnitude Spectrum</p> 
	<p>Original Coefficient : image index : 5 depth 2 index : 3</p> 	<p>Magnitude Spectrum</p> 
3	<p>Original Coefficient : image index : 2 depth 3 index : 3</p> 	<p>Magnitude Spectrum</p> 
	<p>Original Coefficient : image index : 5 depth 3 index : 3</p> 	<p>Magnitude Spectrum</p> 

5. Classification tasks

To test our new Invertible model, we used it for two classification tasks, one is MNIST and the other CIFAR-10.

MNIST

We use 5000 MNIST samples, apply the Invertible Scattering Transform (up to depth 2) on them, and use the coefficients to train a linear classifier. Features are normalized by batch normalization.

We ran the classifier several times, with a batch size of 128 and 20 epochs, and got an accuracy rate of around 97.8% in all runs.

CIFAR-10

In the CIFAR-10 classification we used a hybrid Invertible Scattering Transform + CNN classifier.

The input data first undergoes invertible scattering transforms. The scattering coefficients are then normalized using batch normalization and passed through a feature extractor consisting of convolutional layers (based on a VGG-like architecture).

We ran the classifier using a depth of 2 in the Invertible Scattering Transform, for 90 epochs and got an accuracy rate of around 88% ⁶, which is almost the same as the results of running the same classifier but with the original Kymatio Scattering Transform model.

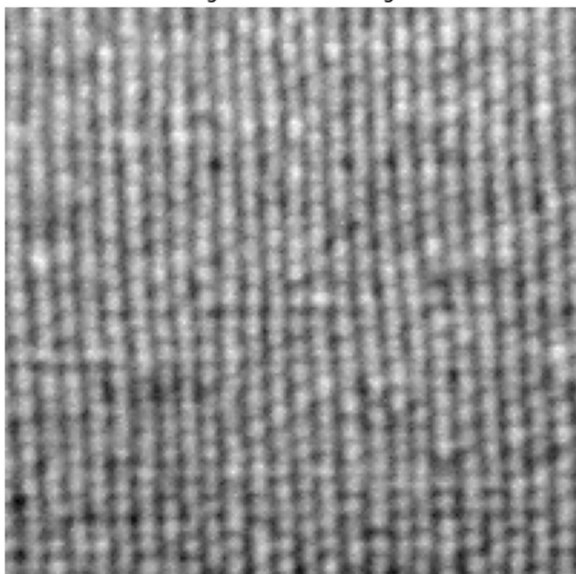
6. Texture reconstruction

Texture reconstruction is a process where using single or multiple samples of a regular pattern as reference, the model tries to generate a new image, having the same kind of pattern.

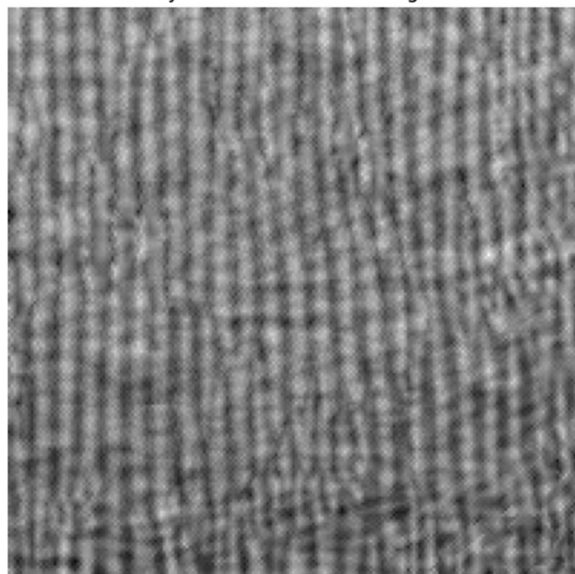
The model we used leverages the Invertible Scattering Transform coefficients to synthesize an image that replicates the statistical and structural properties of the reference texture. By iteratively optimizing the synthesized image to match the reference scattering coefficients, the process achieves a faithful reproduction of the original texture, making it particularly effective for stationary and repetitive patterns.

Below we present the results of running the texture reconstruction model, using the Invertible Scattering Transform on 10 different texture grayscale images taken from the KTH-TIPS dataset.

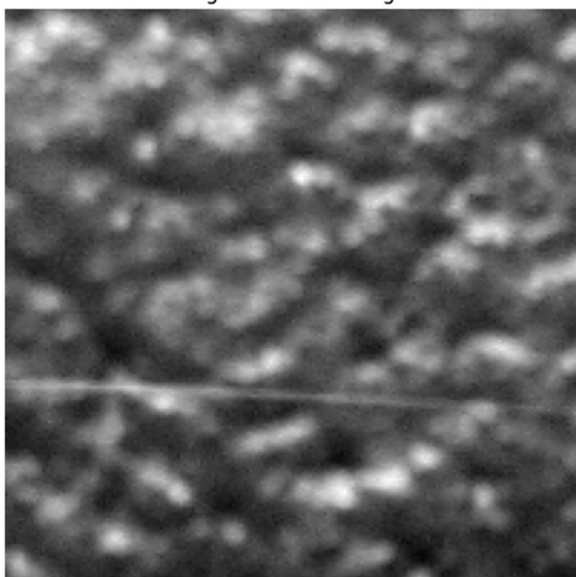
Original Texture Image



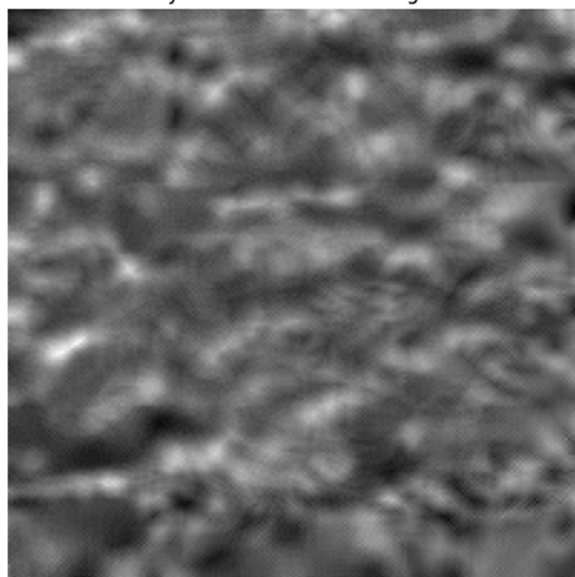
Synthesized Texture Image



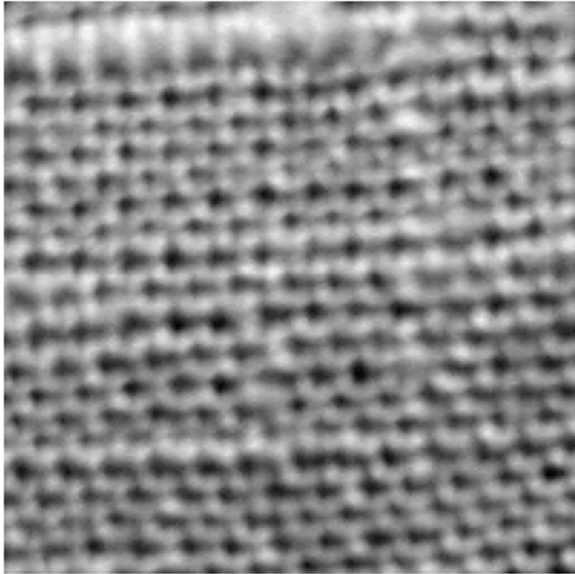
Original Texture Image



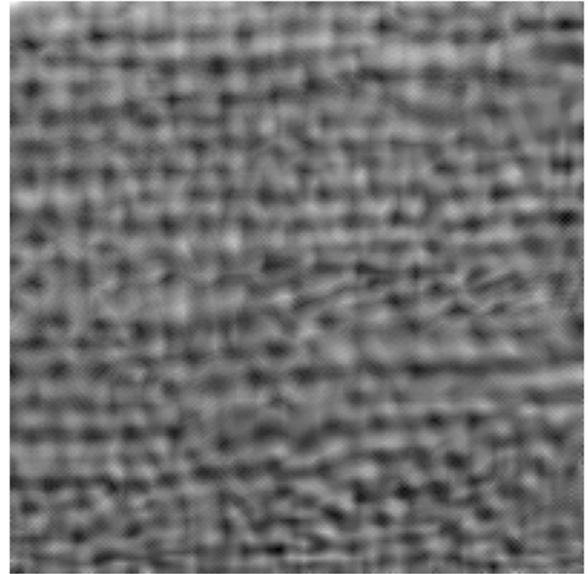
Synthesized Texture Image



Original Texture Image



Synthesized Texture Image

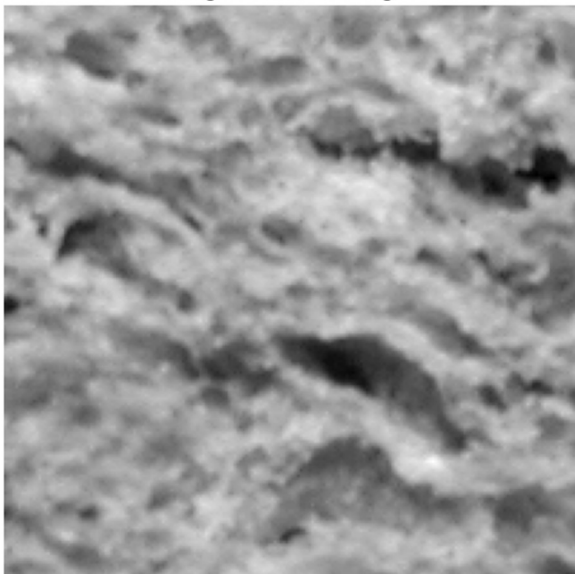


Comparison between regular and invertible Scattering Transform

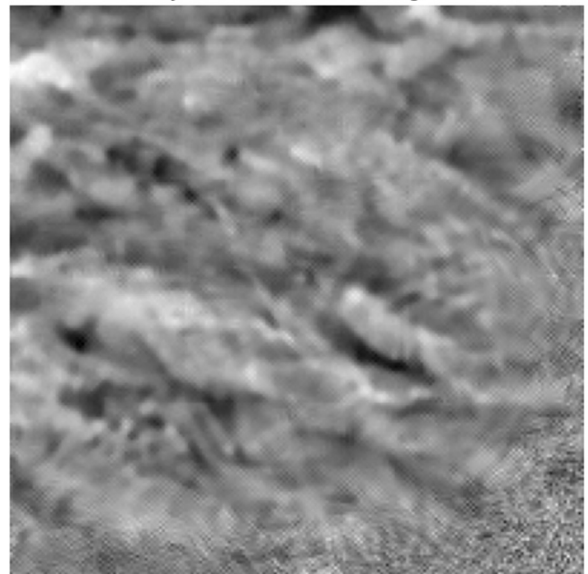
We also ran the same texture reconstruction model, but with the original Kymatio implementation of the Scattering Transform and compared them to our Invertible Scattering Transform.

When running the original Scattering Transform, we got some noise in the lower right corner of the images, which we couldn't really explain.

Original Texture Image

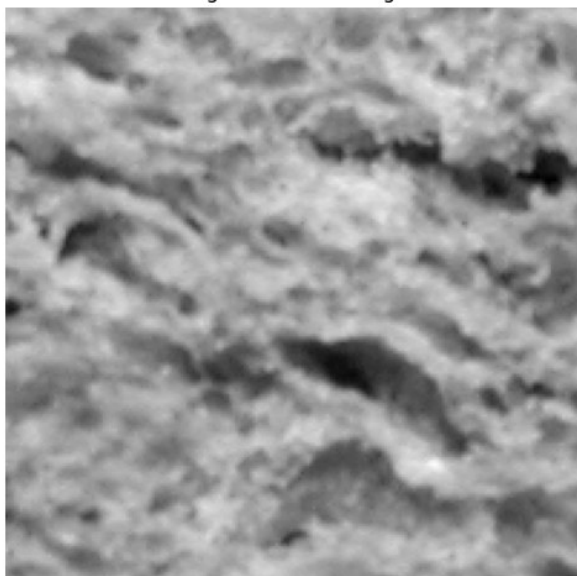


Synthesized Texture Image

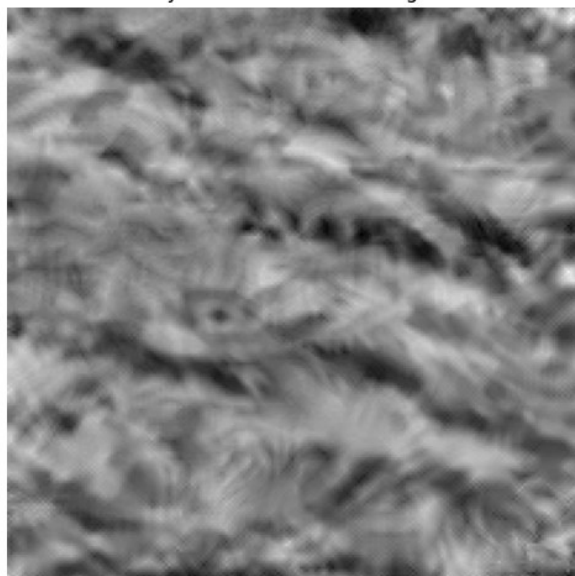


texture reconstruction done with the invertible scattering transform

Original Texture Image



Synthesized Texture Image



texture reconstruction done with the original scattering transform

7. References and links

Levie, Ron. "Convolutional Networks on Grids and Graphs: Mathematical Analysis from a Signal Processing Perspective" .2024. Math. Theory of Convolutional Neural, Technion.

Andreux M., Angles T., Exarchakis G., Leonarduzzi R., Rochette G., Thiry L., Zarka J., Mallat S., Andén J., Belilovsky E., Bruna J., Lostanlen V., Chaudhary M., Hirn M. J., Oyallon E., Zhang S., Cella C., Eickenberg M. (2020). Kymatio: Scattering Transforms in Python. Journal of Machine Learning Research 21(60): 1–6, 2020.

Libraries we used:

Kymatio: <https://github.com/kymatio/kymatio>

scattering: <https://github.com/mariaprat/scattering>

Our code:

<https://github.com/uribh9/invertible-scattering-transform>