Shearlet transforms in Python

March 2022

Contents

Introduction **1**

1 Shearlet transform **2**
1.1 Continouous shearlet system 2
1.2 Cone-adaptated shearlet system 2
1.3 Digital shearlet transform 2

2 Shearlet transform in Python **3**
2.1 Basics 4
2.2 Inverse problems 5
2.2.1 Image denoising 5
2.2.2 Image inpainting 7

Conclusion **8**

Appendix: Codes **9**

Introduction

Shearlets are a framework for the efficient representation of multidimensional data that can capture and encode anisotropic features. Shearlets were first proposed in the mid 2000s and can be considered an extension of wavelets. Both wavelets and shearlets systems are constructed by shifting and dilating (scaling) a locally oscillating generating function. The difference between wavelets and shearlets is in the operator used for scaling. Wavelet transforms are based on isotropic scaling: both dimensions are dilated equally. On the other hand, shearlets use an anisotropic scaling operator that dilates one direction more than the other. In the case of shearlets, a third operator is added: the shear operator which controls the orientation.

In this project, I present the shearlet transform and some examples of simple applications with the dedicated python packages. First, I give the theory of the two-dimensional shearlet systems and transform, in particular the cone-adaptated discrete shearlet systems. Then, I introduce two Python packages that implement the shearlet transform:

`pyShearLab` and `tfShearlab`. I give a tutorial for a basic use of `pyShearLab`, accompanied with visualizations of shearlet systems and coefficients. Then, I implement two simple inverse problem applications in `pyShearLab`: image denoising and inpainting.

1 Shearlet transform

In this section, we introduce the theoretical definitions of shearlet systems and shearlet transforms in continuous and discrete settings. Digital implementations such as the one proposed in `ShearLab3D` and its python counterparts `pyShearLab` and `tfShearlab` are based on cone-adaptated discrete shearlet systems and transforms. The material in this part is directly adapteted from the `ShearLab3D` (MATLAB version) manual [1] and the presentation paper of `tfShearlab` [3].

1.1 Continouous shearlet system

For $\psi \in L^2(\mathbb{R})$, a two-dimensionnal continuous shearlet system is

$$SH_{cont}(\psi) = \{\psi_{a,s,t} = a^{3/4}\psi(A_a^{-1}S_s^{-1}(\cdot - t)) \mid a > 0, s \in \mathbb{R}, t \in \mathbb{R}^2\} \tag{1}$$

where

$$A = \begin{pmatrix} a & 0 \\ 0 & a^{1/2} \end{pmatrix} \qquad\qquad S = \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix}$$

are the *scaling* and *shearing* matrices. The associated shearlet transform of a function $f \in L^2(\mathbb{R})$:

$$f \mapsto \mathcal{SH}_\psi f(a, s, t) = \langle f, \psi_{a,s,t} \rangle \tag{2}$$

1.2 Cone-adaptated shearlet system

Cone-adapted shearlets are introduced in order to resolve issues in capturing horizontally anisotropic structures. The frequency (Fourier) domain is divided into a square-shaped low-frequency part and two conic regions (two vertical cones and two horizontal cones). An illustration is given in Figure 1. The two conic regions are associated with their own shearlet generator functions $\psi, \tilde{\psi}$ $L^2(\mathbb{R})$ and a scaling function $\phi \in L^2(\mathbb{R})$ is introduced for the low-pass region. We define the cone-adaptated continuous shearlet system

$$\mathcal{SH}_{cont}(\phi, \psi, \tilde{\psi}) = \Phi(\phi) \cup \Psi(\psi) \cup \tilde{\Psi}(\tilde{\psi}) \tag{3}$$

where

$$\begin{aligned}
\Phi(\phi) &= \{\phi_t = \phi(\cdot - t) \mid t \in \mathbb{Z}^2\}, \\
\Psi(\psi) &= \{\psi_{a,s,t} = a^{-\frac{3}{4}j}\psi(A_a^{-1}S_s^{-1} \cdot -t) \mid a \in (0,1], |s| \leq 1 + a^{1/2}, t \in \mathbb{Z}^2\}, \\
\tilde{\Psi}(\tilde{\psi}) &= \{\tilde{\psi}_{a,s,t} = a^{-\frac{3}{4}j}\tilde{\psi}(\tilde{A}_a^{-1}S_s^{-T} \cdot -t) \mid a \in (0,1], |s| \leq 1 + a^{1/2}, t \in \mathbb{Z}^2\}
\end{aligned}$$

and with

$$\tilde{A}_a = \text{diag}\,([a^{1/2}, a]).$$

1.3 Digital shearlet transform

The digital shearlet system for digital applications is constructed by sampling continuous shearlet systems on a discrete subset of the parameter space. This yields the discrete cone-adapteted shearlet system.

Cone-adaptated discrete shearlet system Let $\phi, \psi, \tilde{\psi} \in L^2(\mathbb{R}^2)$ and $c = (c_1, c_2) \in \mathbb{R}^2_+$. The cone-adaptated discrete shearlet system $\mathcal{SH}(\phi, \psi, \tilde{\psi}; c)$ is defined by

$$\mathcal{SH}(\phi, \psi, \tilde{\psi}; c) = \Phi(\phi; c_1) \cup \Psi(\psi; c) \cup \tilde{\Psi}(\tilde{\psi}; c),$$

where

$$\begin{aligned}
\Phi(\phi; c_1) &= \{\phi_m = \phi(\cdot - c_1 m) \mid m \in \mathbb{Z}^2\}, \\
\Psi(\psi; c) &= \{\psi_{j,k,m} = 2^{\frac{3}{4}j}\psi(S_k A_{2^j} \cdot - M_c m) \mid j \geq 0, |k| \leq \lceil 2^{j/2} \rceil, m \in \mathbb{Z}^2\}, \\
\tilde{\Psi}(\tilde{\psi}; c) &= \{\tilde{\psi}_{j,k,m} = 2^{\frac{3}{4}j}\tilde{\psi}(S_k^\top \tilde{A}_{2^j} \cdot - \tilde{M}_c m) \mid j \geq 0, |k| \leq \lceil 2^{j/2} \rceil, m \in \mathbb{Z}^2\},
\end{aligned}$$

with the parabolic scaling matrices A_{2^j} and $\tilde{A}_{2^j}$:

$$A_{2^j} = \begin{pmatrix} 2^j & 0 \\ 0 & 2^{j/2} \end{pmatrix} \quad \text{and} \quad \tilde{A}_{2^j} = \begin{pmatrix} 2^{j/2} & 0 \\ 0 & 2^j \end{pmatrix},$$

the shearing matrix S_k:

$$S_k = \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix},$$

and the translation matrices M_c and $\tilde{M}_c$:

$$M_c = \begin{pmatrix} c_1 & 0 \\ 0 & c_2 \end{pmatrix} \quad \text{and} \quad \tilde{M}_c = \begin{pmatrix} c_2 & 0 \\ 0 & c_1 \end{pmatrix}.$$

The shearing matrix adapts the orientation to capture features in different directions. The parabolic scaling matrices changes the scale of the shearlet anisotropically. Thus, anisotropic features can be captured at different scales and orientations. This is illustrated in given in Figure 1.

Cone-adaptated discrete shearlet transform Let $\Lambda = \mathbb{N}_0 \times \{-\lceil 2^{j/2} \rceil, \ldots, \lceil 2^{j/2} \rceil\} \times \mathbb{Z}^2$ be the parameter set. Let $\mathcal{SH}(\phi, \psi, \tilde{\psi}; c)$ be a discrete shearlet system. Then the associated shearlet transform of $f \in L^2(\mathbb{R}^2)$ is:

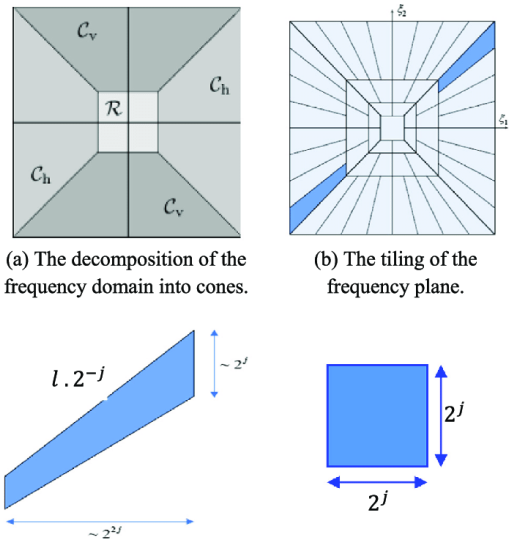
$$f \mapsto \mathcal{SH}(\phi, \psi, \tilde{\psi}) f(m', (j, k, m), (\tilde{j}, \tilde{k}, \tilde{m})) = (\langle f, \phi_{m'} \rangle, \langle f, \psi_{j,k,m} \rangle, \langle f, \tilde{\psi}_{\tilde{j},\tilde{k},\tilde{m}} \rangle) \tag{4}$$

where $((m', (j, k, m), (\tilde{j}, \tilde{k}, \tilde{m})) \in \mathbb{Z}^2 \times \Lambda \times \Lambda$.

2 Shearlet transform in Python

The Python toolbox pyShearLab is adapted from the ShearLab3D toolbox for MATLAB. It offers a set of tools for constructing 2D shearlet systems and appliyng the 2D shearlet transform in Python. There exists an other Python implementation of ShearLab3D based on Tensorflow and Shearlab.jl (the ShearLab3D toolbox for Julia): `tfShearlab` [3, 2]. The latter version computes a tensor shearlet transform using the tensor representations of Tensorflow. It is stated in the presentation article of `tfShearlab` [3] that it outperforms previous libraries in several experiments including image denoising and inpainting.

In this section, I give an illustrated tutorial on how to use `pyShearLab`. First, I introduce the basics: how to construct a 2D shearlet system and visualize it, the shearlet transform, the plot of coefficients and the inverse shearlet transform. Then, I reproduce and adapt some of the experiments presented in [3]: image denoising and inpainting.



(a) The decomposition of the frequency domain into cones.

(b) The tiling of the frequency plane.

(c) The frequency support of the shearlet.

(d) The frequency support of the wavelet.

Figure 1: Illustration of the decomposition of the frequency domain and the tilling of the frequency plane, from [5].

2.1 Basics

In `pyShearLab`, a (cone-adaptated discrete) shearlet system can be constructed with the function `SLgetShearletSystem2D`.

```
shearletSystem = pyshearlab.SLgetShearletSystem2D(GPU, rows, cols, nScales)
```

where `rows, cols` are the dimensions of the data, `nScales` is the number of scales to compute and `useGPU` is a boolean. The i-th resulting shearlet of the system is stored in

```
shearletSystem['shearlets'].real[:, :, i]
```

A plot of a shearlet system for a 512×512 grid, 2 scales and shearing parameters $\{-1, 1, 0\}$ computed by `SLgetShearletSystem2D` is represented in the Figure 2. One can easily see the the conic frequency representation and the effect of the shearing and scaling parameters. The associated shearlet of a grey-scale (2D) image `X` is computed with the `pyshearlab.SLsheardec2D` function:

```
coeffs = pyshearlab.SLsheardec2D(X, shearletSystem)
```

and is illustrated in the Figure 4. One can see the the effect of the shearing parameter and scale parameter, as well as the horizontal (top) and vertical (bottom) cone structure. Reconstruction is achieved with the `SLshearrec2D` function:

```
Xrec = pyshearlab.SLshearrec2D(coeffs, shearletSystem)
```

2.2 Inverse problems

In this part, I present two inverse problem applications of the shearlet transform: shearlet-based image denoising and shearlet-based inpainting.

2.2.1 Image denoising

The principle of shearlet-based image denoising is to apply a hard thresholding to the shearlet coefficients of the noisy image, and then to reconstructed the image by the inverse shearlet transform. Let $X \in \ell^2(\mathbb{Z})$ be a gray-scale image, to which we apply Gaussian white noise. Let X_{noise} be the resulting image, that is:

$$X_{noise}[i,j] = X[i,j] + \epsilon[i,j],$$

where $\epsilon[i,j] \backsim \mathcal{N}(0,\sigma^2)$. We compute

$$X_{denoised} = \mathcal{SH}^{-1}\mathcal{T}_\delta \mathcal{SH} X_{noise},$$

where $\mathcal{T}_\delta$ the hard thresholding operator given by

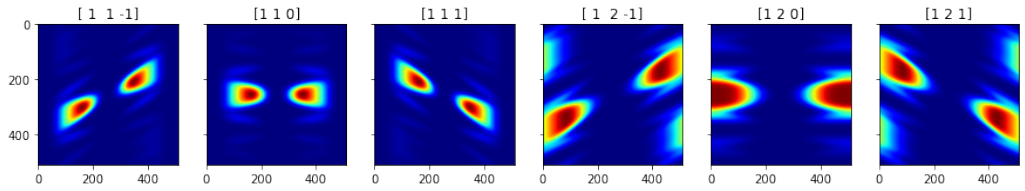
$$(\mathcal{T}_\delta x)(n) := \begin{cases} x(n) & \text{if } |x[n]| \geq \delta, \\ 0 & \text{else.} \end{cases}$$

As recommended in [3], we use a threshold that varies with the scale:

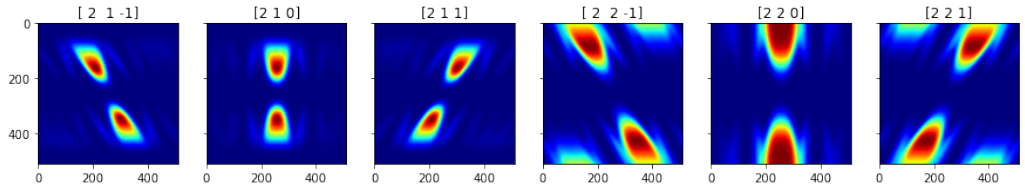
$$\delta_j = K_j \sigma.$$

The corresponding implementation with `pyShearLab` is given in the following code:

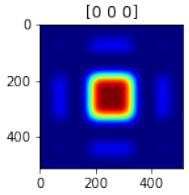
```
import numpy as np

#Apply Gaussian white noise to the image X
sigma = 30
Xnois = X + sigma * np.random.randn(X.shape[0], X.shape[1])

# Generate shearlet system with 4 scales (e.g.)
shearletSystem = pyshearlab.SLgetShearletSystem2D(0, X.shape[0], X.shape
    [1], 4)

# Decomposition
coeffs = pyshearlab.SLsheardec2D(Xnois, shearletSystem)

thresholdingFactor = 3

#Thresholding weights [RMS are root mean squares (L2-norm divided by
#sqrt(X.shape[0]*X.shape[1])) of the system shearlets]
weights = np.ones(coeffs.shape)
for j in range(len(shearletSystem["RMS"])):
    weights[:,:,j] = shearletSystem["RMS"][j]*np.ones((X.shape[0], X.shape
    [1]))

#Hard thresholding
thresh_coeffs = np.real(coeffs)
zero_indices = np.abs(coeffs) / (thresholdingFactor * weights * sigma) < 1
thresh_coeffs[zero_indices] = 0

# Reconstruction
Xrec = pyshearlab.SLshearrec2D(thresh_coeffs, shearletSystem)

#Compute PSNR
PSNR = pyshearlab.SLcomputePSNR(X, Xrec)
```



(a) Horizontal cones

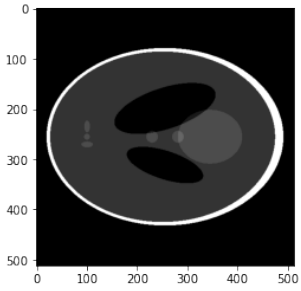


(b) Vertical cones

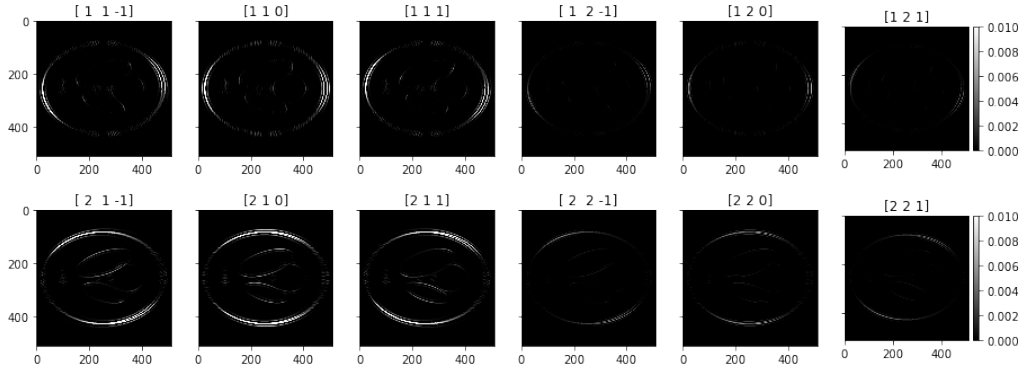


(c) Low frequency region

Figure 2: A shearlet system with 2 scales and shearing $= \{-1, 1, 0\}$, for an image size of 512×512. The shearlets are specified in the format `[cone scale shearing]`.

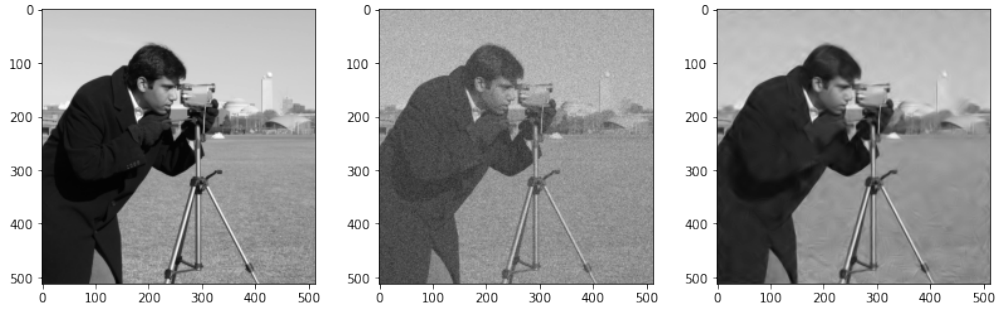


(a) Shepp–Logan phantom gray-scale 512×512 image.



(b) Shearlet coeffcients of the 2D shearlet transform of the Shepp–Logan phantom.

Figure 3: 2D shearlet transform of Shepp–Logan phantom, for the shearlet system in Figure 2.



(a) Original (b) Gaussian white noise added (c) Denoised, PSNR=28.4

Figure 4: Image denoising based on hard thresholding of the shearlet coefficients of the noisy image.

2.2.2 Image inpainting

Let $X \in \ell^2(\mathbb{Z}^2)$ be a gray-scale image that was partially occluded by a binary mask $M \in \{0,1\}^{\mathbb{Z}\times\mathbb{Z}}$, that is:

$$X_{masked}[i,j] = X[i,j]M[i,j].$$

I adapt and apply the algorithm proposed in [3] which is based on iterative hard thresholding on subsequent shearlet transforms. This algorithm is given below:

Algorithm 3 Shearlet inpainting by iterative hard thresholding

Input: $f_{masked}, M, \delta_{init}, \delta_{min}, iterations$
Output: $f_{inpainted}$

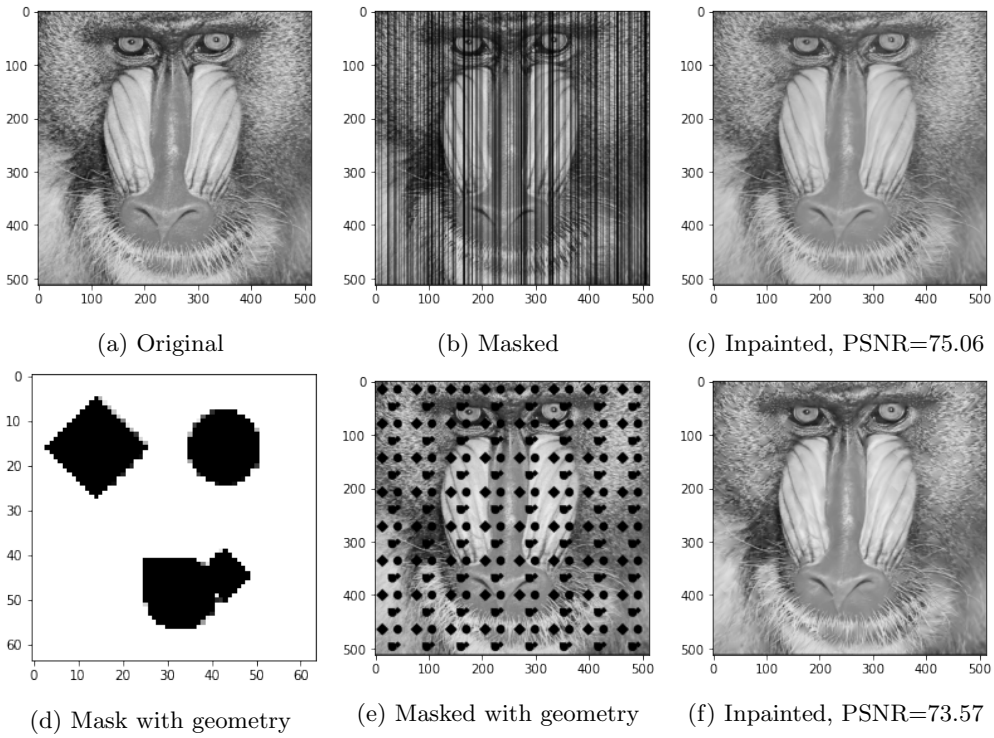
$f_{inpainted} := 0$;
$\delta := \delta_{init}$;
$\lambda := (\delta_{min})^{1/(iterations-1)}$
for *i:=1 to iterations do* **do**
  $f_{res} := M \cdot (f_{masked} - f_{inpainted})$;
  $f_{inpainted} := SH^{-1}\mathcal{T}_\delta SH(f_{res} + f_{inpainted})$;
  $\delta := \lambda\delta$;
end

The corresponding implementation in `pyShearLab`

```
def inpaint2D(XMasked, mask, nIter,stopFactor,shearletSystem):
    # Compute coefficients
    coeffs = pyshearlab.SLsheardec2D(XMasked, shearletSystem)
    # Normalize the coefficients
    coeffsNormalized = np.zeros(coeffs.shape, np.complex64)
    for i in range(shearletSystem['nShearlets']):
        coeffsNormalized[:,:,i] = coeffs[:,:,i] / shearletSystem['RMS'][i]
    # Define the parameters
    delta = np.abs(coeffsNormalized).max()
    lbda = (stopFactor)**(1/(nIter-1))
    XInpainted = np.zeros(XMasked.shape)
    #Iterative thresholding
    for it in range(nIter):
        res = mask * (XMasked - XInpainted);
        coeffs = pyshearlab.SLsheardec2D(XInpainted+res, shearletSystem);
        coeffsNormalized = np.zeros(coeffs.shape, np.complex64);
        for i in range(shearletSystem['nShearlets']):
            coeffsNormalized[:,:,i] = coeffs[:,:,i]/shearletSystem['RMS'][i
    ]
        coeffs = coeffs * (np.abs(coeffsNormalized) > delta)
        XInpainted = pyshearlab.SLshearrec2D(coeffs, shearletSystem)
        delta = delta * lbda
        print('Iteration ', it, ' of ', nIter)
    return XInpainted
```

I create masks (random vertical lines and a mask with geometric objects) and I apply the inpainting algorithm to the masked 512×512 gray-scale images. I use the same parameters (number of iterations and stopFactor) that are used in [3], and a shearlet system with 4 scales. The resulst are shown in Figure 5.

```
#Inpainting
nIter = 150
stopFactor = 0.005
Xinpainted = inpaint2D(Xmasked, mask, nIter, stopFactor, shearletSystem)
```



(a) Original (b) Masked (c) Inpainted, PSNR=75.06

(d) Mask with geometry (e) Masked with geometry (f) Inpainted, PSNR=73.57

Figure 5: Image inpainting based on iterative shearlet transform and hard thresholding.

Conclusion

With this project, I discovered the shearlet transform, which allows to encode efficiently the anisotropic features of multidimensional data. I was able to implement applications of the shearlet transform to two inverse problems: denoising and inpainting, using `pyShearLab`. Unfortunately, I was not able to use the `tfShearlab` package because of installation problems. The tensor representation used in `tfShearLab` increases the efficiency of the shearlet transform algorithms and their implementation in the TensorFlow architecture allows their integration in deep networks. Later on, it would be interesting to investigate applications with deep networks like the one developped in [4].

References

[1] Shearlab 3d manual v1.0. 2013. http://shearlab.math.lmu.de/files/documents/ShearLab3Dv10%5FManual%2Epdf.

[2] Héctor Andrade Loarca. tfshearlab. https://github.com/arsenal9971/tfshearlab.

[3] Héctor Andrade Loarca and Gitta Kutyniok. tfshearlab: The tensorflow digital shearlet transform for deep learning. *arXiv preprint arXiv:2006.04591*, 2020. https://arxiv.org/pdf/2006.04591.pdf.

[4] Kutyiniok Gitta Andrade-Loarca, Héctor and Ozan Öktem. Shearlets as feature extractor for semantic edge detection: The model-based and data-driven realm. In *arXiv preprint: arXiv:1911.12159*, 2019.

[5] Heba Elhoseny, Wael El-Rahman, Walid El-Shafai, Ghada El Banby, El-Sayed El-Rabaie, Fathi Abd El-Samie, Osama Faragallah, and Korany Mahmoud. Efficient multi-scale non-sub sampled shearlet fusion system based on modified central force optimization and contrast enhancement. *Infrared Physics Technology*, 102, 07 2019.