Report - APL405 - GROUP 10
PINN in image processing

Hemali Priyadarshi
Ishaan Govil
Revanth Vasireddi
Darshan Rakhewar
Nisha Saha

April 19, 2022

Contents

1 Introduction

Image denoising is the technique of removing noise or distortions from an image.First, we shall point out the drawbacks of the the conventional methods of image denoising using the isotropic heat equation. Following will be the motivation to introduce the Perona-Malik model (Anisotropic heat equation) and the issue it resolves. Later, this paper will present our proposed solution of performing image denoising by visualising it as a physical model guided by PDE's and solving them using Finite Difference method (FDM) and Physics Informed Neural Networks (PINN).

This method of solving the non-linear differential equation transforms the image smoothing process into solving an optimization problem that can be solved by a physics informed neural network. The image is interpreted as a 2-dimensional array of pixels which would be passed as an input to the neural network along with the time variable and then using the boundary and the initial conditions we would train the model to output the processed image.

In the final analysis, we have reported some numerical experiments to show the effectiveness of the proposed machine learning based approach for image denoising.

2 Problem statement

The conventional method used for image denoising have been well known to dilute even the important details of the image like the edges and important features that differentiate them from the rest of the image and help us identify them.
This well known methods for the same is -
Making use of the *Heat equation* to model the diffusion process where in place of the temperature gradient we have the intensity gradient and using a constant diffusion coefficient. Then solving this PDE we would get an output function $u = u(x, y, t)$ which models the intensity of the pixel at spatial co-ordinates (x, y) and at time t. So the differential equation we are solving in this method is -

$$u_t = div(c \times \nabla u) \tag{1}$$

where c is a constant.
The solution of this isotropic equation is the Gaussian kernel.
Using a Gaussian Kernel to convolve the input image so as to filter out the unwanted noise in the image. However, since the kernel is uniform hence it does not take into account the intensity difference present at difference points (which indicates an edge) and performs uniform blurring throughout.

Let's first discuss this method of Isotropic diffusion in more detail to motivate the need for the anisotropic diffusion.

3 Isotropic Diffusion Method

Linear diffusion (heat) equation solves the image denoising problem as illustrated below -
Let u_0 be a noisy image in the domain Ω, With the initial condition as shown below -

$$u(x, y, 0) = u_0(x, y), \qquad t\epsilon[0, T] \tag{2}$$

this equation defines an isotropic diffusion model for image denoising. The evolving image $u(x, y, t)$ *where* $t\epsilon[0, T]$ is a denoised image at the time t. On solving, this is equivalent to performing a Gaussian smooth filtering, which is a fundamental operation for enhancing digital images. If the image u_0 is square integrable, then the explicit solution of the differential equation is given by

$$u(x, y, t) = (G_{\sqrt{2t}} * u_0)(x, y) \tag{3}$$

where $G_\sigma(x, y)$ denotes the 2-D Gaussian kernel, and the operation $*$ is the convolution operator.
This heat equation removes the noise, but it generates sequentially more and more blurred images with each iteration, also blurring the features of importance such as edges leading to a barely recognizable image.

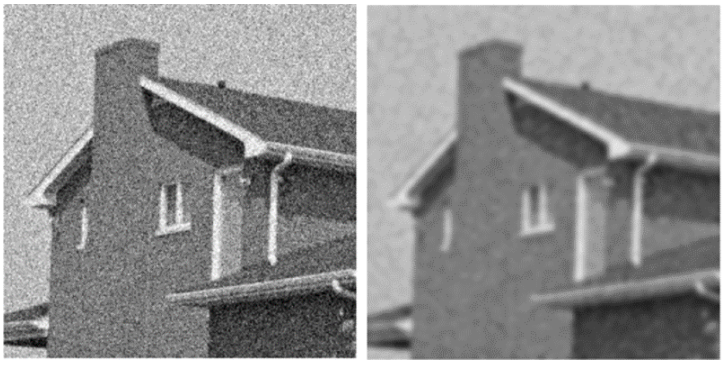

Figure 1: Image denoised with isotropic heat equation

As is evident from the example shown above, the main drawback of the isotropic diffusion is that the smoothing can damage image features such as edges and lines. To avoid such damages, the amount and the direction of smoothing needs to be controlled. To overcome this drawback, Perona and Malik derived a nonlinear extension of the heat equation, leading to anisotropic diffusion.

4 The Anisotropic diffusion or the Perona-Mallik Model

Perona-Mallik suggested an anisotropic diffusion filtering of the image which acts as a low pass filter. This technique aims at reducing image noise without removing significant parts of the image content, typically edges, lines or other details that are important for the interpretation of the image.
Perona-Mallik coefficient (c) controls the rate of diffusion and has been chosen as a function of the signal gradient so as to preserve edges in the image.

$$\frac{\partial u}{\partial t} = \nabla c.\nabla u + c(x,t)\Delta u \tag{4}$$

The diffusivity constant c depends on space activity in the given part of a picture, measured by the norm of the local pixel intensity gradient given as-

1 Inside the homogeneous region of the image where the magnitude of the gradients is small, the above equation should perform isotropic smoothing for removing the noise.

2 Near the feature boundaries in the image, the smoothing is stopped due to a large value of intensity gradient, and sharp details are thus preserved in the restored image.

Some example diffusion coefficient functions that have been suggested by Perona-Mallik are -

$$g(\nabla I) = \exp\left(\frac{-||\nabla I||}{K}\right)^2 \tag{5}$$

$$g(\nabla I) = \frac{1}{1 + \left(\frac{-||\nabla I||}{K}\right)^2} \tag{6}$$

In our study we would be making use of the second function which is a small scale approximation of the first function itself in the limit that the gradient is very small.

Figure 2: Comparison of noisy image with isotropic and anisotropic solutions

In the above comparison it is vivid that when the isotropic solution is used then the resulting image is denoised but has lost sharp features due to blurring of edges while these features are preserved with anisotropic solution due to piece-wise denoising of only the homogeneous regions. So to counter this problem of uniform smoothening we would be making use of a non-constant diffusion coefficient function and then using the PINN model to find an approximate solution to the modified non-linear differential equation.

5 FDM

5.1 Introduction

The Perona-Malik model with fractional derivatives and its application for image processing. This model is obtained from the standard PM equation by replacing the ordinary derivative with a fractional derivative. Hence the Perona-Mallik equation in a simplified manner can be represented as follows below-

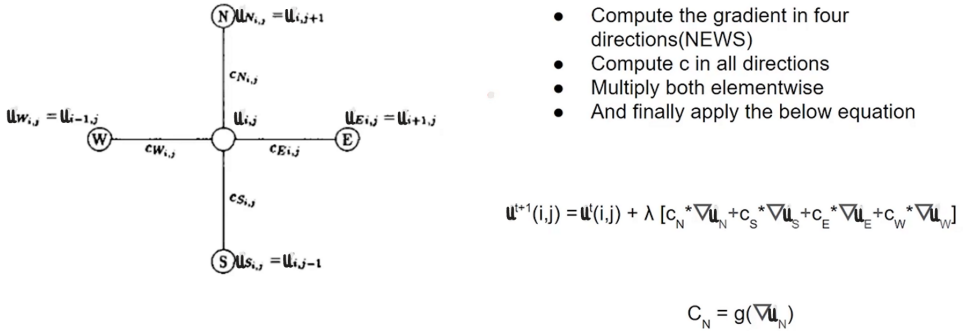

Figure 3: FDM arrangement of Perona-Malik diffusion. (Here the pixel intensity at a certain time instant and at position (i,j) is represented as $u^t_{i,j}$

$$g(\nabla I) = \exp\left(\frac{-||\nabla I||}{K}\right)^2 \tag{7}$$

$$g(\nabla I) = \frac{1}{1 + \left(\frac{-||\nabla I||}{K}\right)^2} \tag{8}$$

5.2 Mathematical Derivation

:

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(g\left(\frac{\partial u}{\partial x}\right) \times \left(\frac{\partial u}{\partial x}\right)\right) + \frac{\partial}{\partial y}\left(g\left(\frac{\partial u}{\partial y}\right) \times \frac{\partial u}{\partial y}\right)\right) \tag{9}$$

$$\frac{\Delta u}{\Delta t} = \frac{\Delta}{\Delta x}\left(g\left(\frac{\Delta u}{\Delta x}\right) \times \left(\frac{\Delta u}{\Delta x}\right)\right) + \frac{\Delta}{\Delta y}\left(g\left(\frac{\Delta u}{\Delta y}\right) \times \left(\frac{\Delta u}{\Delta y}\right)\right) \tag{10}$$

Using the forward difference at time t and a second-order central difference for the space derivative at position $u_{i,j}$ (FTCS), we get the recurrence equation:

$$\frac{u_{i,j}^{t+1} - u_{i,j}^{t}}{\Delta t} = \frac{g(\frac{u_{i+1,j}^{t} - u_{i,j}^{t}}{\Delta x}) \times \frac{u_{i+1,j}^{t} - u_{i,j}^{t}}{\Delta x} - g(\frac{u_{i,j}^{t} - u_{i-1,j}^{t}}{\Delta x}) \times \frac{u_{i,j}^{t} - u_{i-1,j}^{t}}{\Delta x}}{\Delta x} +$$
$$\frac{g(\frac{u_{i+1,j}^{t} - u_{i,j}^{t}}{\Delta y}) \times \frac{u_{i+1,j}^{t} - u_{i,j}^{t}}{\Delta y} - g(\frac{u_{i,j}^{t} - u_{i-1,j}^{t}}{\Delta y}) \times \frac{u_{i,j}^{t} - u_{i-1,j}^{t}}{\Delta y}}{\Delta y}$$

$$u_{i,j}^{t+1} = u_{i,j}^{t} + \Delta t \times [g(\frac{u_{i+1,j}^{t} - u_{i,j}^{t}}{\Delta x}) \times \frac{u_{i+1,j}^{t} - u_{i,j}^{t}}{\Delta x^2} - g(\frac{u_{i,j}^{t} - u_{i-1,j}^{t}}{\Delta x}) \times \frac{u_{i,j}^{t} - u_{i-1,j}^{t}}{\Delta x^2} +$$
$$g(\frac{u_{i+1,j}^{t} - u_{i,j}^{t}}{\Delta y}) \times \frac{u_{i+1,j}^{t} - u_{i,j}^{t}}{\Delta y^2} - g(\frac{u_{i,j}^{t} - u_{i-1,j}^{t}}{\Delta y}) \times \frac{u_{i,j}^{t} - u_{i-1,j}^{t}}{\Delta y^2}]$$

Considering $\Delta x \approx \Delta y \approx 1unit$ and $\frac{\Delta t}{\Delta x^2} = \lambda$

So, with this recurrence relation, and knowing the values at time t, one can obtain the corresponding values at time t+1.

We can obtain $u_{i,j}^{t+1}$ from the other values this way:

$$u_{i,j}^{t+1} = u_{i,j}^{t} + \lambda \times [g(u_{i+1,j}^{t} - u_{i,j}^{t}) \times (u_{i+1,j}^{t} - u_{i,j}^{t}) - g(u_{i,j}^{t} - u_{i-1,j}^{t}) \times (u_{i,j}^{t} - u_{i-1,j}^{t})] + [g(u_{i+1,j}^{t} - u_{i,j}^{t}) \times (u_{i} \tag{11}$$

The numerical resolution of this model is based on the explicit scheme of finite difference method. **Boundary condition**: $\nabla u_{i,j}^{t+1}.\hat{n} = 0$

$$\text{replacing the following } \left\{ \begin{array}{ll} u_{i,j}^{t+1} = u_{i+1,j}^{t+1} & i = 0 \\ u_{i,j}^{t+1} = u_{i-1,j}^{t+1} & i = -1 \\ u_{i,j}^{t+1} = u_{i,j+1}^{t+1} & j = 0 \\ u_{i,j}^{t+1} = u_{i,j-1}^{t+1} & j = -1 \end{array} \right\}$$

5.3 Implementation

:

```
    # The main diffusion function which implements the FDM

def anisodiff(im, steps, b, lam = 0.25):
    #takes image input,
    # the number of timesteps,
    # the hyperparameter 'b' and lambda
```

```
tolerance=0.001
im_temp= np.pad(im, [(1,1), (1,1)], mode='constant')

#Add extra boundary with same pixels
# as th boundary for easier computation of gradients
im_temp[0,1:-1]=im[0,:]
im_temp[-1,1:-1]=im[-1,:]
im_temp[1:-1,0]=im[:,0]
im_temp[1:-1,-1]=im[:,-1]
im_new=im

for t in range(steps):

    dn = im_temp[:-2,1:-1] - im_temp[1:-1,1:-1]
    # [a0,a1,a2,a3] - [a1,a2,a3,a4]
    # -- taking finite difference of the consecutive elements
    # backside difference
    # (del(u)/del(x))-

    ds = im_temp[2:,1:-1] - im_temp[1:-1,1:-1]
    # [a2,a3,a4,a5] - [a1,a2,a3,a4]
    #  -- taking finite difference of the consecutive elements
    # frontside difference for change in x
    # (del(u)/del(x))+

    de = im_temp[1:-1,2:] - im_temp[1:-1,1:-1]
    # [a2,a3,a4,a5] - [a1,a2,a3,a4]
    # -- taking finite difference of the consecutive elements
    # frontside difference for change in y
    # (del(u)/del(y))+

    dw = im_temp[1:-1,:-2] - im_temp[1:-1,1:-1]
    # [a0,a1,a2,a3] - [a1,a2,a3,a4]
    # -- taking finite difference of the consecutive elements
    # frontside difference for change in y
    # (del(u)/del(x))-
```

```
        im_new = (im_temp[1:-1,1:-1] + \
            lam * (f(dn,b)*dn + f (ds,b)*ds + f (de,b)*de + f (dw,b)*dw))
        # The perona malik equation implementation
        # In each step more and more blurring will
        # take place leading to removal
        # of more and more noise from the image.
        #im (the original image) is modified in each step

        #boundary condition i.e grad(u).n=0
        im_new[0,:]=im_new[1,:]
        im_new[-1,:]=im_new[-2,:]
        im_new[:,0]=im_new[:,1]
        im_new[:,-1]=im_new[:,-2]

        diff=im_new-im

        #stop flow when there is no significant
        #change at successive time-steps
        error=np.sum(np.square(100*diff[:]))/np.sum(np.square(im_new))
        flag=0
        if error<tolerance :
            if flag==0:
                pyp.imshow(im_new, cmap='gray')
            flag=1
            break

        im=im_new
        #update the dummy image
        im_temp[1:-1, 1:-1] = im
        im_temp[0,1:-1]=im[0,:]
        im_temp[-1,1:-1]=im[-1,:]
        im_temp[1:-1,0]=im[:,0]
        im_temp[1:-1,-1]=im[:,-1]

    return im

#  ****(Diffusivity)Edge-Stopping Functions*****
```

```
def f(grad_u,b):
    return np.exp(-1* (np.power(grad_u,2))/(np.power(b,2)))
##the diffusion coefficient is going to be input as b

def f2(grad_u,b):
func = 1/(1 + ((grad_u/b)**2))
return func
```

5.4 Input , Output Specifications

Thus Finite Difference Method (FDM) is modeled programmatically in the code as follows -

Input of the anisotropic diffusion function is im (as 2D tensor having corresponding pixel intensities), steps (number of iterations to be made), b (The regularization parameter 'K') and lam(FDM parameter lambda).
Output im (as 2D tensor having corresponding pixel intensities) after the specified number of FDM steps for PM diffusion.
Process:

- Here dn, de, ds, dw are the gradients on the image intensities at the point in the directions North, East, South and West respectively.
- In every iteration, the stopping function is calculated for the obtained values of the gradients in accordance with the relation shown in Figure 6.
- Finally, im (the image) is modified according to the equation in Figure 5.

5.5 Results - FDM

The sample input and output shown above clearly indicates that the Perona-mallik differential equation can successively be solved using the Finite difference method thus performing denoising without destroying the edge details.

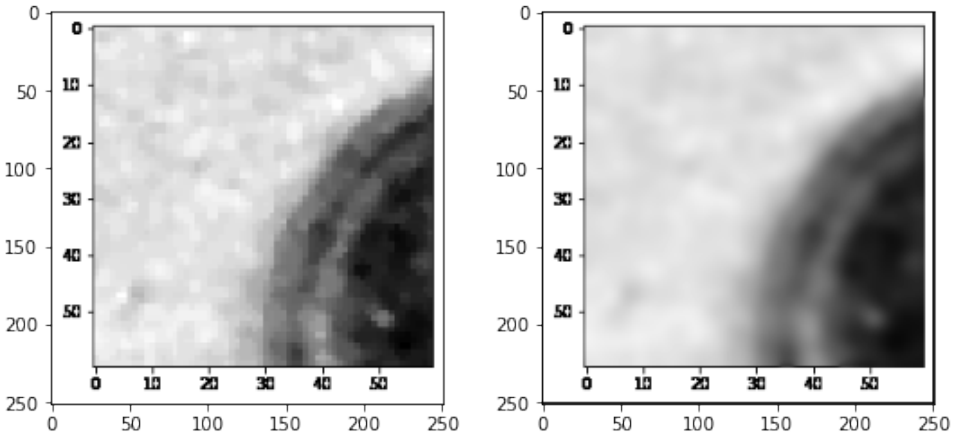

Figure 4: Input and output result of FDM implementation

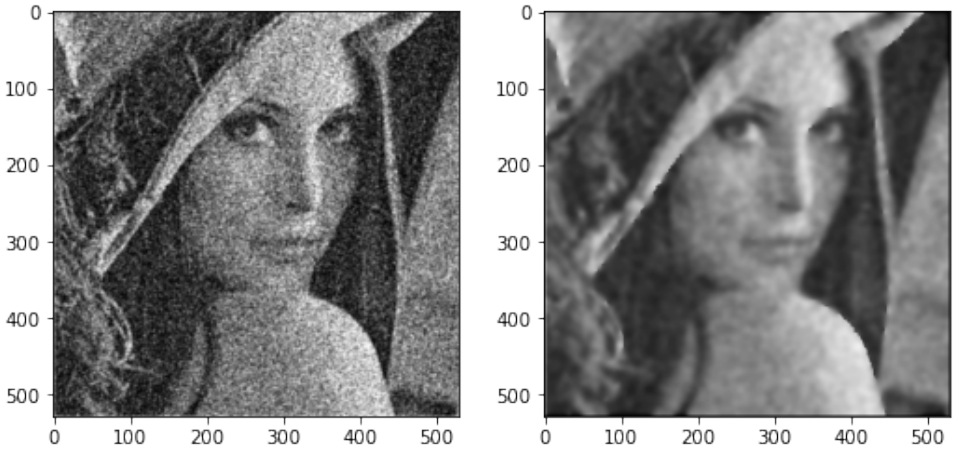

Figure 5: Another input and output result of FDM implementation

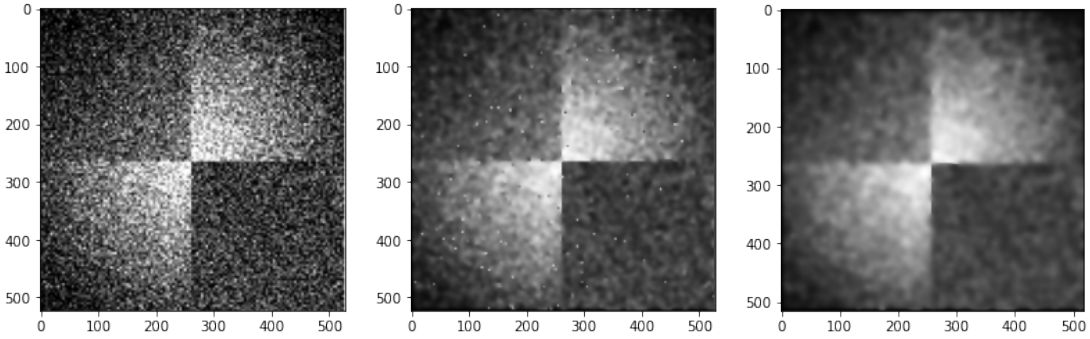

Figure 6: Input and output of denoising an image successively two times

6 PINN Model

Now, we are going to try implementing a neural network to obtain the smoothed image.

6.1 Overview of PINN

Physics-informed neural networks (PINNs) leverage governing physical equations in neural network training i.e. PINNs are designed to be trained to satisfy the given training data as well as the imposed governing equations.

Potentially, an accurate solution of partial differential equations can be found without knowing the boundary conditions. Therefore, with some knowledge about the physical characteristics of the problem and some form of training data (even sparse and incomplete), PINN may be used for finding an optimal solution with high fidelity.

PINNs allow for addressing a wide range of problems in computational science and represent a pioneering technology leading to the development of new classes of numerical solvers for PDEs. In our project we would be making use of PINN's to solve the well-known Perona-Malik equation and making use of it in the field of image processing.

6.2 Cost Functions of PINN

There are three types of losses for the problem in hand (g is the diffusion co-efficient function, n is the normal to the boundries of the image) -

- **Differential Equation Loss**

$$\sum_{x,y\in\Omega,t>0} (\frac{\partial u(x,y,t)}{\partial t} - div[g(|\nabla u(x,y,t)|)\nabla u(x,y,t)])^2 \tag{12}$$

- **Initial Condition Loss**

$$\sum_{x,y\in\Omega} (u(x,y,0) - u_0)^2 \tag{13}$$

- **Boundary Condition Loss**

$$\sum_{x\in\{0,x_{size}-1\},y\in\{0,y_{size}-1\},t>0} (\frac{\partial u(x,y,t)}{\partial n})^2 \tag{14}$$

6.3 Input/Output Specifications

The **inputs** to our model consists of X , Y and T where :-
Input of the neural network are 3- 4D arrays covering each permutation of (x, y, t) as its element when viewed at different indices of size $(n \times n \times m)$, where m represented the no of time steps at which we want to discretize the time frame , and assuming that the image is represented as an array of pixels of size $(n \times n)$,

where the corresponding index entries at a particular index represent one specific pixel in space (specific x and y) at a particular instant of time(specific t).

Output is the ndarray containing each predicted u(x, y, t) for each corresponding (x, y, t) at the same index and is fount using the f.eval(x_input, y_input, t_input) function once the model has been trained.

6.4 Implementation using sciann

For implementing the *PINN* model using *SciANN* the following code has been used -

6.4.1 Setting up Neural Network

```
x = sn.Variable('x')
y = sn.Variable('y')
t = sn.Variable('t')
f = sn.Functional('f', [x, y, t], [10, 20, 20, 10], 'tanh')
```

6.4.2 Implementation of losses

Note :

sn.math.diff (f, x, order = 1) gives $\frac{\partial f}{\partial x}$
sn.math.sign(x) is 1 if $x > 0$, 0 if $x = 0$, -1 if $x < 0$

- **Differential Equation Loss**

```
## defining the loss/objective function
f_x = sn.math.diff(f, x, order=1)
f_y = sn.math.diff(f, y, order=1)
f_t = sn.math.diff(f, t, order=1)
```

```
mod_deltax = f_x/(1+((f_x*f_x)+(f_y*f_y))/1000)
mod_deltay = f_y/(1+((f_x*f_x)+(f_y*f_y))/1000)

# divergence of (g(s). del u)
div = sn.math.diff(mod_deltax, x, order=1)
div = div + sn.math.diff(mod_deltay, y, order = 1)

# differential equation loss -> equation loss for t>0
L2 = (f_t - div)**2
```

- **Boundary Loss**

```
# boundary loss
TOL = 0.001
C2 = (1 - sn.math.sign(x - (TOL))) * (f_x * f_x) #x = 0
# x = last
C3 = (1 + sn.math.sign(x - (image.shape[0] - TOL))) * (f_x * f_x)

C4 = (1 - sn.math.sign(y - (TOL))) * (f_y * f_y) #y = 0
# y = last
C5 = (1 + sn.math.sign(y - (image.shape[1] - TOL))) * (f_y * f_y)
```

- **Initial Loss**

  As you can see in the model below, d1 is taken as the data of f and d2 is taken as the data of Total_loss. These are made to get equal to initial image and zero respectively. Thus, the initial loss is taken care of, when we give an array containing actual image in the parameter while training.

6.4.3 Defining the model

```
Total_loss = C1 + C2 + C3 + C4 + C5 + L2

## defining the model
d1 = sn.Data(f)
d2 = sn.Data(Total_loss)
m = sn.SciModel([x, y, t], [d1, d2])
```

6.4.4 Training the Model

To train the model we make use of SciANN.train function which takes as arguments –

- The SciModel defined above (m)
- The input data i.e *x_data y_data and t_data*
- The original image, so as to calculate the initial losses.
- The learning rate
- The no of epochs/iterations to be done

6.5 Results of PINN

The following examples have been worked out for **100** iterations and the time interval considered is **10** units.

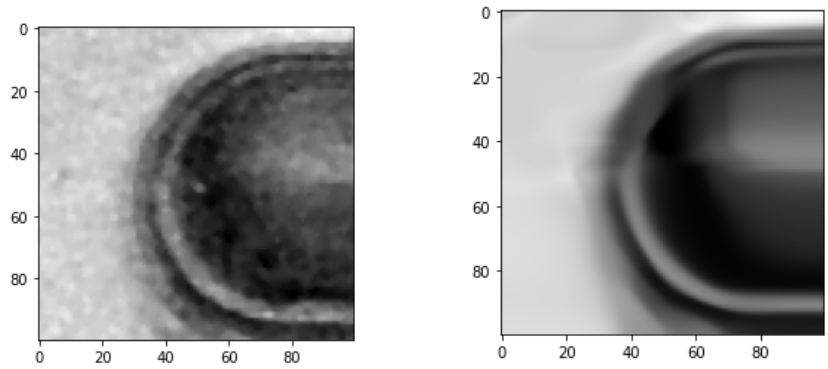

Figure 7: Image and Smoothed Image

7 Comparision of Results

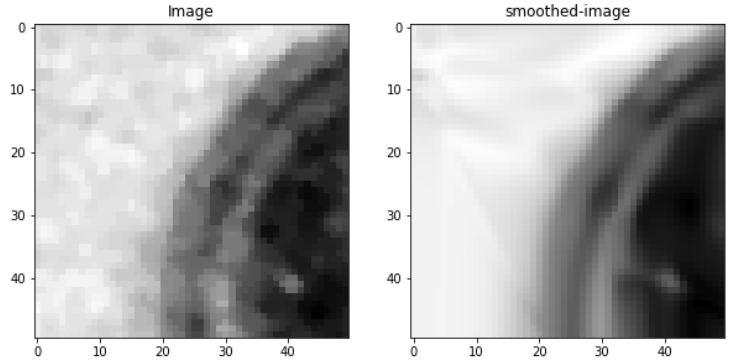

Figure 8: PINN Model output for test image-1

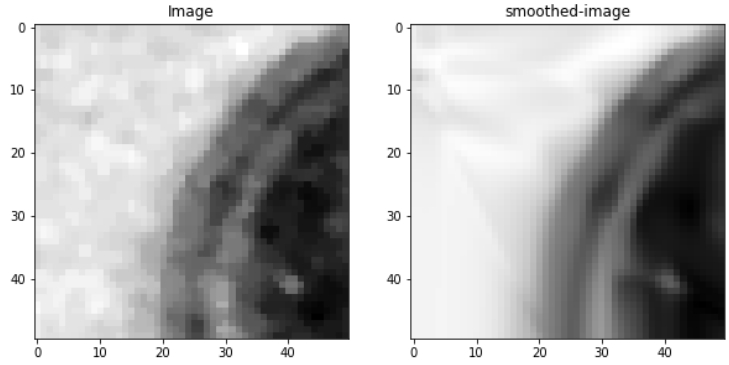

Figure 9: FDM output for test image-2

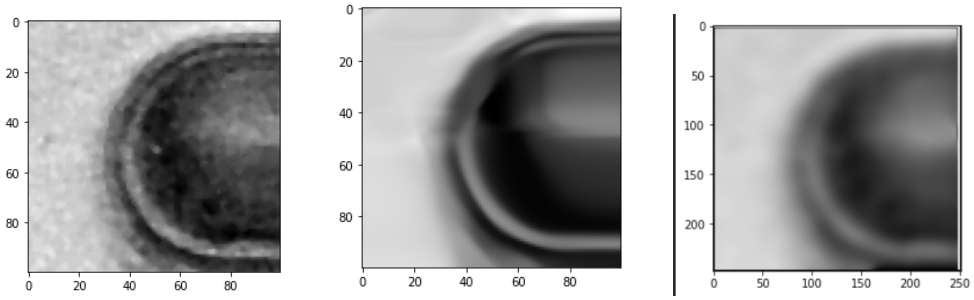

Figure 10: Original image, PINN output and FDM output

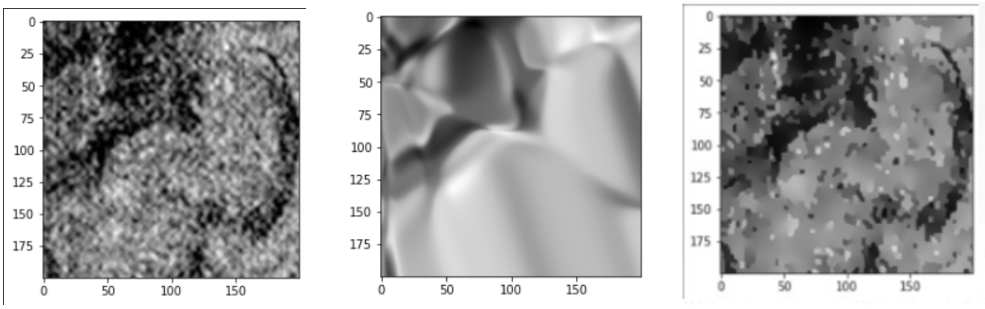

Figure 11: Original image, PINN output and FDM output

8 Conclusion

As can be seen from the above results, the PINN model is working better than the FDM implementation as it is giving a more cleaner and less noisy image. Also it takes care of the edges as we can see that the diffusion taking place at the edges is much less as compared to other parts of the image. This was expected when using anisotropic diffusion in our PINN model.

The conclusion of the above paper is that it is possible to visualize and solve the evolution of images governed by heat equations by applying physics-informed neural networks for solving nonlinear diffusion equation modeled for removing additive noise of digital images.

Further steps -

- While acquisition and transmission of images, all recording devices have physical limitations and traits which make them prone to noise.
- Noise manifests itself in the form of signal perturbation leading to deterred image observation, image analysis and image assessment. Image denoising is fundamental to the world of image processing.
- Thus any progress made in image denoising forms a stepping stone in the understanding of image processing and statistics.
- To extend our model to even more detailed images we would need to implement multithreading as well as make use of the Graphical Processing Unit so that the operations taking place our faster and the no of iterations we can run our model on also increases thus producing better results.

References

- https://arxiv.org/abs/1412.6291v1 - Perona-Malik equation and its numerical properties
- https://arxiv.org/abs/2202.00595v1 - Adaptive Finite Element Method for Image Processing
- https://github.com/pastapleton/Perona-Malik/blob/master/Perona-Malik%20Ivan.py
- A least squares support vector regression for anisotropic diffusion filtering - Arsham Gholamzadeh Khoee , Kimia Mohammadi Mohammadi , Mostafa Jani , Kourosh Parand