IMAGE SMOOTHING USING ANISOTROPIC DIFFUSION FILTER

TEAM NAME: KKR
Tanya Chanchalani

PROBLEM STATEMENT

Adaptive denoising process by means of anisotropic diffusion filtering.

MOTIVATION

Denoising an image with linear filters such as Gaussian filters is conceptually easy, however, the image details and edges are degraded, resulting in a blurry denoised image. But the anisotropic nonlinear diffusion method reduces noise in flat regions and preserves edges to a greater degree.

THEORY

Diffusion is a physical process which aims at minimizing differences in the spatial concentration u(x,t) of a substance.

The process can be described by two equations :

- Fick's law : $j = -g\nabla u$ (j is diffusion flux , g is diffusivity constant and ∇u is concentration gradient)
- Continuity equation : $\partial_t u = -\operatorname{div} j$

This leads to the diffusion equation $\partial_t u = \operatorname{div}(g \cdot \nabla u)$

THEORY

$\partial_t u = \mathrm{div}\,(g \cdot \nabla u)$ in this equation:

- If g=1 it is known as isotropic or linear diffusion
- If g is a function of u then it is called non linear diffusion
- If g is matrix-valued then it leads to process where diffusion is different in different directions. This is called anisotropic diffusion
- Perona-Malik model leads to less smoothing in locations of strong edge information.
- We use gradient norm as an edge detector, diffusivity should decrease with increasing with $|\nabla u|$.

$$g(|\nabla u|) = \frac{1}{\sqrt{1 + |\nabla u|^2/\lambda^2}}$$

THEORY

Perona and Malik (1987) formulated NLDF(Non linear diffusion filter) in which they replaced the diffusion constant g in the equation t u = div (g · ∇u) with a scalar valued function of gradient ∇u of the grey levels in the image. The diffusion equation then reads

$$\partial_t u = div\left(g\left(|\nabla u|\right)\nabla u\right)$$

CODE

```
[ ]  import numpy as np
     import warnings
     import cv2

     def anisodiff(img,niter=1,kappa=50,gamma=0.1,step=(1.,1.),option=1,ploton=True):
         """
         Anisotropic diffusion.

         Usage:
         imgout = anisodiff(im, niter, kappa, gamma, option)

         Arguments:
                 img    - input image
                 niter  - number of iterations
                 kappa  - conduction coefficient 20-100 ?
                 gamma  - max value of .25 for stability
                 step   - tuple, the distance between adjacent pixels in (y,x)
                 option - 1 Perona Malik diffusion equation No 1(high contrast over low contrast)
                          2 Perona Malik diffusion equation No 2(wider regions over smaller ones)
                 ploton - if True, the image will be plotted on every iteration

         Returns:
                 imgout   - diffused image.

         kappa controls conduction as a function of gradient.

         gamma controls speed of diffusion

         step is used to scale the gradients
         """
```

CODE

```
if img.ndim == 3:
    warnings.warn("Only grayscale images allowed, converting to 2D matrix")
    img = img.mean(2)

# initialize output array
img = img.astype('float32')
imgout = img.copy()

# initialize some internal variables
deltaS = np.zeros_like(imgout)
deltaE = deltaS.copy()
NS = deltaS.copy()
EW = deltaS.copy()
gS = np.ones_like(imgout)
gE = gS.copy()

# create the plot figure, if requested
if ploton:
    import pylab as pl
    from time import sleep

    fig = pl.figure(figsize=(20,5.5),num="Anisotropic diffusion")
    ax1,ax2 = fig.add_subplot(1,2,1),fig.add_subplot(1,2,2)

    ax1.imshow(img,interpolation='nearest')
    ih = ax2.imshow(imgout,interpolation='nearest',animated=True)
    ax1.set_title("Original image")
    ax2.set_title("Iteration 0")

    fig.canvas.draw()
```

CODE

```
for ii in range(niter):

    # calculate the diffs
    deltaS[:-1,: ] = np.diff(imgout,axis=0)
    deltaE[: ,:-1] = np.diff(imgout,axis=1)

    # conduction gradients (only need to compute one per dim!)
    if option == 1:
        gS = np.exp(-(deltaS/kappa)**2.)/step[0]
        gE = np.exp(-(deltaE/kappa)**2.)/step[1]
    elif option == 2:
        gS = 1./(1.+(deltaS/kappa)**2.)/step[0]
        gE = 1./(1.+(deltaE/kappa)**2.)/step[1]

    # update matrices
    E = gE*deltaE
    S = gS*deltaS

    # subtract a copy that has been shifted 'North/West' by one pixel
    NS[:] = S
    EW[:] = E
    NS[1:,:] -= S[:-1,:]
    EW[:,1:] -= E[:,:-1]

    # update the image
    imgout += gamma*(NS+EW)
```

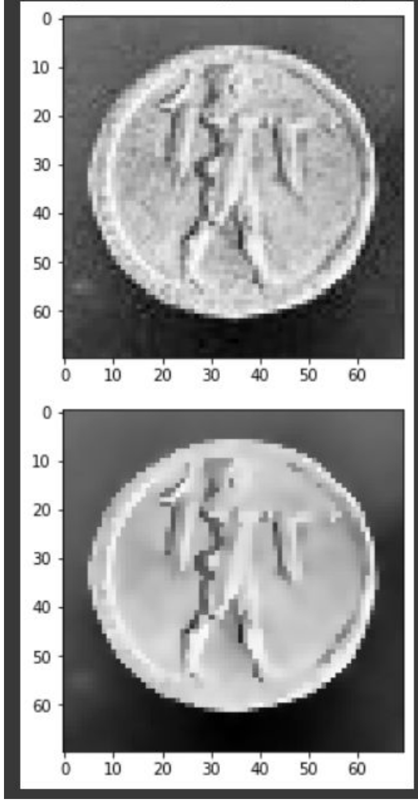
CODE

```
    if ploton:
        iterstring = "Iteration %i" %(ii+1)
        ih.set_data(imgout)
        ax2.set_title(iterstring)
        fig.canvas.draw()
        # sleep(0.01)

return imgout
```

```
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
#from scipy.misc import imread
#im = imread("lenna.png")
from matplotlib.pyplot import imread
#im =coins[10:80, 300:370]
#im = lenna()[..., 0]
result = anisodiff(im, niter=10, kappa=25)
plt.imshow(im, cmap="Greys_r")
plt.figure()
plt.imshow(result, cmap="Greys_r")
```

CODE

```
<matplotlib.image.AxesImage at 0x7f9ddfa0d510>
```



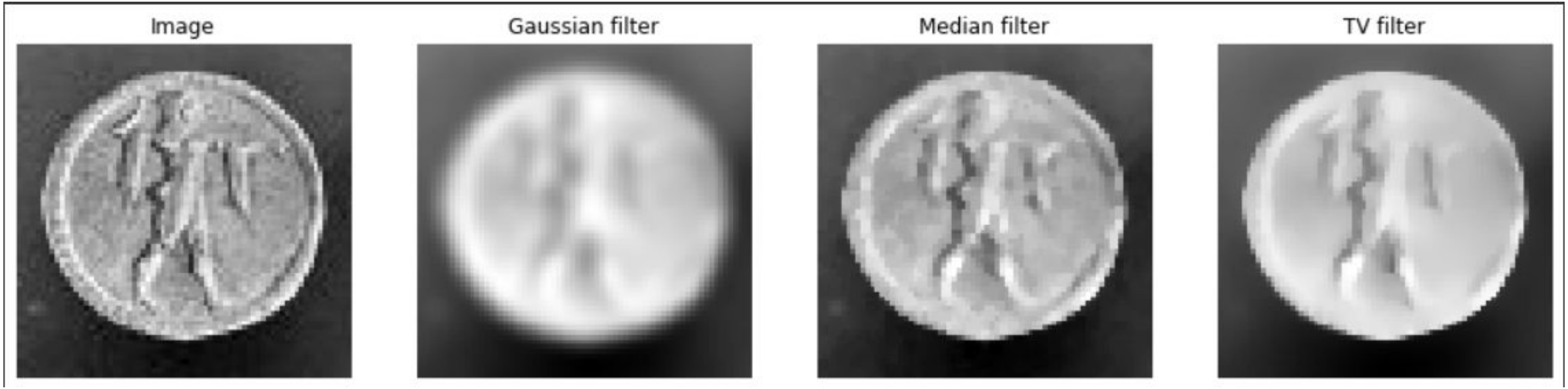
CODE

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data
from skimage import filters
from skimage import restoration

coins = data.coins()
gaussian_filter_coins = filters.gaussian(coins, sigma=2)
med_filter_coins = filters.median(coins, np.ones((3, 3)))
tv_filter_coins = restoration.denoise_tv_chambolle(coins, weight=0.1)
```

CODE

```
plt.figure(figsize=(16, 4))
plt.subplot(141)
plt.imshow(coins[10:80, 300:370], cmap='gray', interpolation='nearest')
plt.axis('off')
plt.title('Image')
plt.subplot(142)
plt.imshow(gaussian_filter_coins[10:80, 300:370], cmap='gray',
           interpolation='nearest')
plt.axis('off')
plt.title('Gaussian filter')
plt.subplot(143)
plt.imshow(med_filter_coins[10:80, 300:370], cmap='gray',
           interpolation='nearest')
plt.axis('off')
plt.title('Median filter')
plt.subplot(144)
plt.imshow(tv_filter_coins[10:80, 300:370], cmap='gray',
           interpolation='nearest')
plt.axis('off')
plt.title('TV filter')
plt.show()
```

CODE



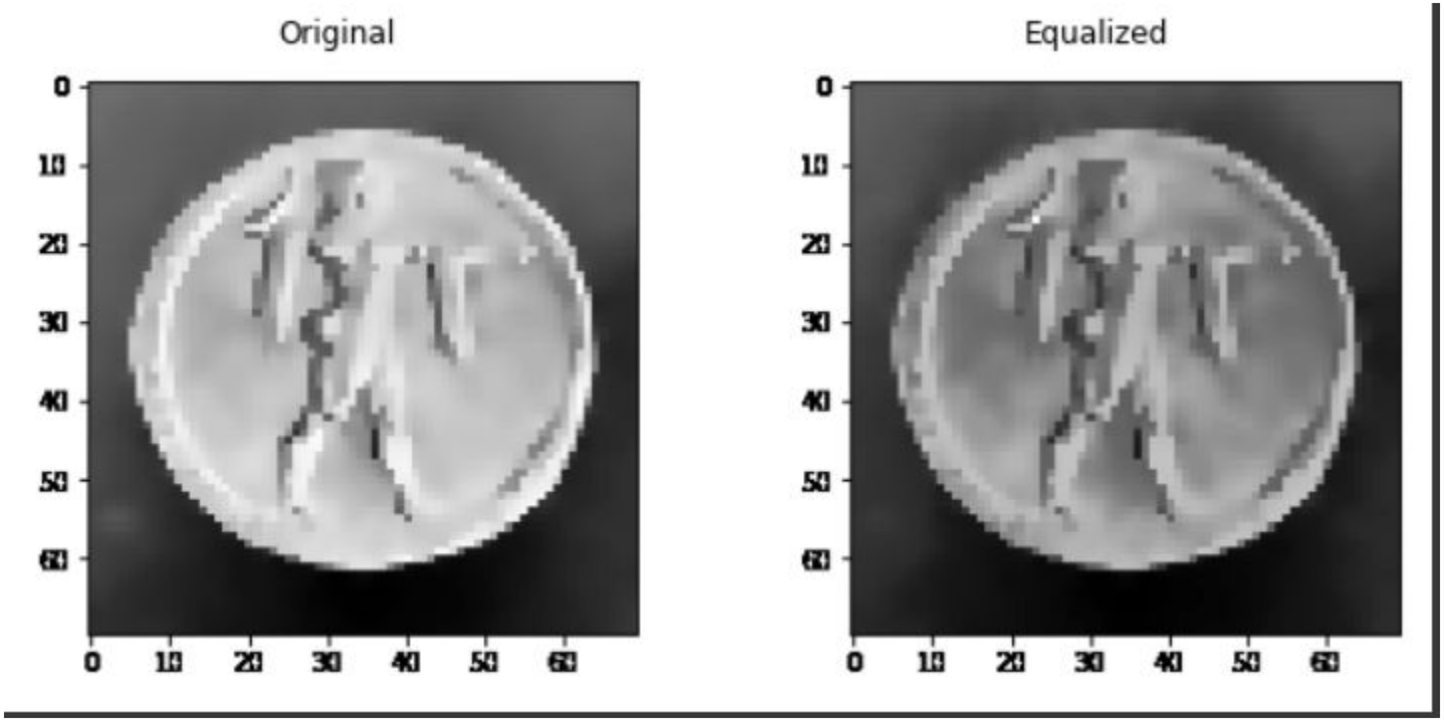
CODE

```
import numpy as np
import cv2
import json
from matplotlib import pyplot as plt

def read_this(image_file, gray_scale=False):
    image_src = cv2.imread(image_file)
    if gray_scale:
        image_src = cv2.cvtColor(image_src, cv2.COLOR_BGR2GRAY)
    else:
        image_src = cv2.cvtColor(image_src, cv2.COLOR_BGR2RGB)
    return image_src
```

CODE

```
def equalize_this(image_file, with_plot=False, gray_scale=False):
    image_src = read_this(image_file=image_file, gray_scale=gray_scale)
    if not gray_scale:
        r_image, g_image, b_image = cv2.split(image_src)

        r_image_eq = cv2.equalizeHist(r_image)
        g_image_eq = cv2.equalizeHist(g_image)
        b_image_eq = cv2.equalizeHist(b_image)

        image_eq = cv2.merge((r_image_eq, g_image_eq, b_image_eq))
        cmap_val = None
    else:
        image_eq = cv2.equalizeHist(image_src)
        cmap_val = 'gray'

    if with_plot:
        fig = plt.figure(figsize=(10, 20))

        ax1 = fig.add_subplot(2, 2, 1)
        ax1.axis("off")
        ax1.title.set_text('Original')
        ax2 = fig.add_subplot(2, 2, 2)
        ax2.axis("off")
        ax2.title.set_text("Equalized")

        ax1.imshow(image_src, cmap=cmap_val)
        ax2.imshow(image_eq, cmap=cmap_val)
        return True
    return image_eq
equalize_this(image_file='coins.png', with_plot=True)
```

CODE



CODE

```
import cv2
import numpy as np

x=cv2.imread('coins.png')
y=cv2.imread('coinsbb.png')
def compute_psnr(img1, img2):
    img1 = img1.astype(np.float64) / 255.
    img2 = img2.astype(np.float64) / 255.
    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
        return "Same Image"
    return 10 * math.log10(1. / mse)
print(compute_psnr(x,y))
```

CODE

```
import cv2
import numpy as np
import math
z=cv2.imread('coins.png')
r=cv2.imread('gaussb.png')
def compute_psnr(img1, img2):
    img1 = img1.astype(np.float64) / 255.
    img2 = img2.astype(np.float64) / 255.
    mse = np.mean((img1 - img2) ** 2)
    if mse == 0:
        return "Same Image"
    return 10 * math.log10(1. / mse)
print(compute_psnr(z,r))
```

CODE

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
img= cv2.imread('coins.png')
img1=cv2.imread('coinsbb.png')
images=np.concatenate((img,img1),axis=1)
cv2_imshow(images)
cv2.waitKey(0)
cv2.destroyAllWindows()
gray_img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
gray_img1=cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
hist=cv2.calcHist(gray_img,[0],None,[256],[0,256])
hist1=cv2.calcHist(gray_img1,[0],None,[256],[0,256])
plt.subplot(121)
plt.title("Image1")
plt.xlabel('bins')
plt.ylabel("No of pixels")
plt.plot(hist)
plt.subplot(122)
plt.title("Image2")
plt.xlabel('bins')
plt.ylabel("No of pixels")
plt.plot(hist1)
plt.show()
```

CODE

```
gray_img_eqhist=cv2.equalizeHist(gray_img)
gray_img1_eqhist=cv2.equalizeHist(gray_img1)
hist=cv2.calcHist(gray_img_eqhist,[0],None,[256],[0,256])
hist1=cv2.calcHist(gray_img1_eqhist,[0],None,[256],[0,256])
plt.subplot(121)
plt.plot(hist)
plt.subplot(122)
plt.plot(hist1)
plt.show()
eqhist_images=np.concatenate((gray_img_eqhist,gray_img1_eqhist),axis=1)
cv2_imshow(eqhist_images)
cv2.waitKey(0)
cv2.destroyAllWindows()
clahe=cv2.createCLAHE(clipLimit=40)
gray_img_clahe=clahe.apply(gray_img_eqhist)
gray_img1_clahe=clahe.apply(gray_img1_eqhist)
images=np.concatenate((gray_img_clahe,gray_img1_clahe),axis=1)
cv2_imshow(images)
cv2.waitKey(0)
cv2.destroyAllWindows()
th=80
max_val=255
```

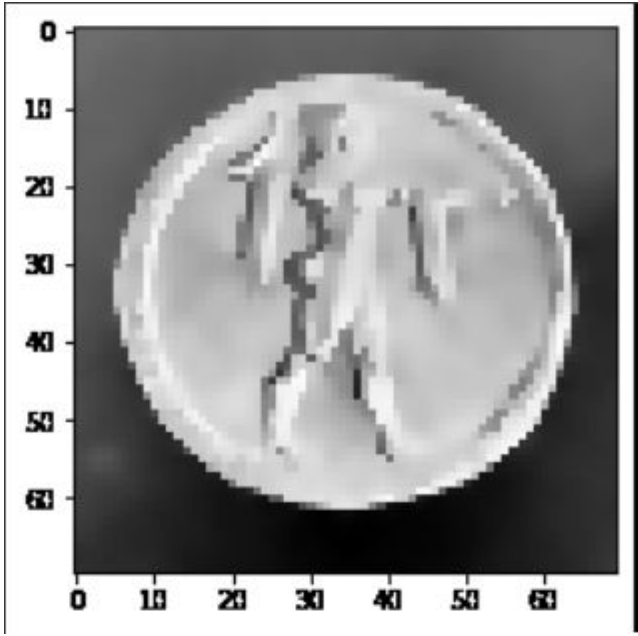
CODE

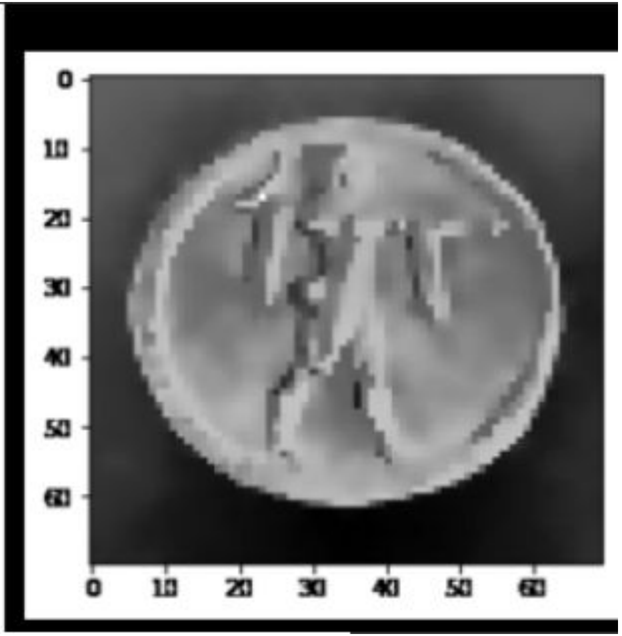
```
ret, o1 = cv2.threshold(gray_img_clahe, th, max_val, cv2.THRESH_BINARY)
cv2.putText(o1,"Thresh_Binary",(40,100),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),3,cv2.LINE_AA)
ret, o2 = cv2.threshold(gray_img_clahe, th, max_val, cv2.THRESH_BINARY_INV)
cv2.putText(o2,"Thresh_Binary_inv",(40,100),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),3,cv2.LINE_AA)
ret, o3 = cv2.threshold(gray_img_clahe, th, max_val, cv2.THRESH_TOZERO)
cv2.putText(o3,"Thresh_Tozero",(40,100),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),3,cv2.LINE_AA)
ret, o4 = cv2.threshold(gray_img_clahe, th, max_val, cv2.THRESH_TOZERO_INV)
cv2.putText(o4,"Thresh_Tozero_inv",(40,100),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),3,cv2.LINE_AA)
ret, o5 = cv2.threshold(gray_img_clahe, th, max_val, cv2.THRESH_TRUNC)
cv2.putText(o5,"Thresh_trunc",(40,100),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),3,cv2.LINE_AA)
ret ,o6=  cv2.threshold(gray_img_clahe, th, max_val,  cv2.THRESH_OTSU)
cv2.putText(o6,"Thresh_OSTU",(40,100),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),3,cv2.LINE_AA)
```

CODE

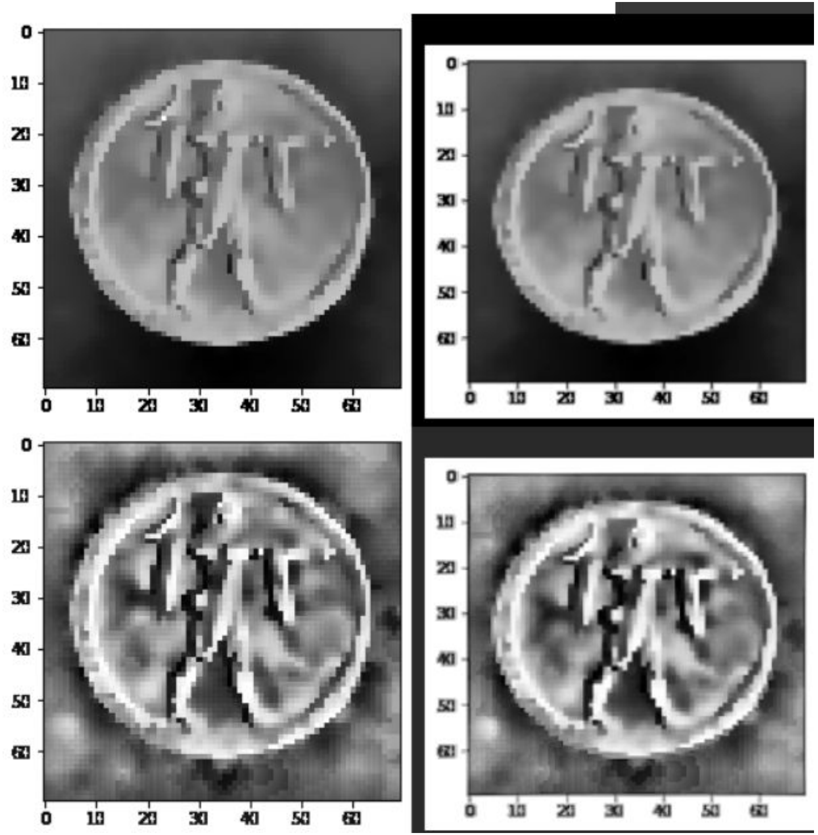
```
final=np.concatenate((o1,o2,o3),axis=1)
final1=np.concatenate((o4,o5,o6),axis=1)

cv2.imwrite("Image1.jpg",final)
cv2.imwrite("Image2.jpg",final1)
gray_image = cv2.imread('coins.png',0)
gray_image1 = cv2.imread('coinsbb.png',0)
thresh1 = cv2.adaptiveThreshold(gray_image, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2)
thresh2 = cv2.adaptiveThreshold(gray_image, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 31, 3)
thresh3 = cv2.adaptiveThreshold(gray_image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 13, 5)
thresh4 = cv2.adaptiveThreshold(gray_image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 31, 4)
thresh11 = cv2.adaptiveThreshold(gray_image1, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2)
thresh21 = cv2.adaptiveThreshold(gray_image1, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 31, 5)
thresh31 = cv2.adaptiveThreshold(gray_image1, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 21,5 )
thresh41 = cv2.adaptiveThreshold(gray_image1, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 31, 5)

final=np.concatenate((thresh1,thresh2,thresh3,thresh4),axis=1)
final1=np.concatenate((thresh11,thresh21,thresh31,thresh41),axis=1)
cv2.imwrite('rect.jpg',final)
cv2.imwrite('rect1.jpg',final1)
gray_image = cv2.imread('coins.png',0)
gray_image1 = cv2.imread('coinsbb.png',0)
ret,thresh1 = cv2.threshold(gray_image,0, 255,  cv2.THRESH_BINARY+cv2.THRESH_OTSU)
ret,thresh2 = cv2.threshold(gray_image1,0, 255,  cv2.THRESH_BINARY+cv2.THRESH_OTSU)

cv2.imwrite('rect.jpeg',np.concatenate((thresh1,thresh2),axis=1))
```

CODE


0
10
20
30
40
50
60
0
10
20
30
40
50
60


0
10
20
30
40
50
60
0
10
20
30
40
50
60

CODE



FUTURE SCOPE

Anisotropic methods of diffusion degrades the fine structure and reduces the resolution of the image. In future we can use various quality enhancement methods and adaptive learning filters.

It is clear, however, that nonlinear diffusion filtering is a young field which has certainly not reached its final state yet.

CONCLUSION

As the performed denoising experiments and the method comparison provided encouraging results.

The anisotropic diffusion technique gives better enhanced image than Gaussian filter and this result is backed by SNR. Because of this method's strong edge-preserving character, it can be successfully used for edge detection or computer vision tasks like object detection, satellite image enhancement.

REFERENCES

1)”On the choice of the parameters for anisotropic diffusion in image processing,” C. Tsiotsios, M. Petrou.

2)”Adaptive Image Denoising Method Based on Diffusion Equation and Deep Learning,” S.Ma, C.Zhang, Journal of Robotics, vol.2022, Article ID 7115551, 9 pages, 2022.

3) “Variational methods for computer vision,” Prof.Dr. Daniel Cremers.

Q&A

THANK YOU