Labsheet 9
Reducing high frequency noise

Ragja Palakkadavath
(SC18M003)

Sub: Image and Video Processing Lab
Department: Mathematics(MLC)
Date of Lab sheet: October 04, 2018
Date of Submission: October 16, 2018

1. Anisotropic diffusion

Add noise to your image and do anisotropic diffusion.
Refer the book Image Processing by Petrou, page 349

Aim

To perform anisotropic diffusion to a noised image

Theory/Discussion

Anisotropic diffusion is a technique aiming 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. It is an algorithm that generalises Gaussian filtering, used to reduce additive Gaussian noise to make it adaptive to local image gradient, so that edges are preserved.

Algorithm

1. Read an Image
2. Find the value for B using the histogram of the magnitude vectors of the image. Starting from the first bin, accumulate the entries of the successive bins until C% of pixels have been accounted for. The value of the last bin is noted as threshold B
3. Find the $\delta N, \delta S, \delta E, \delta W$ values for every pixel in the image.
4. Following that, find $c_K(i,j) = g(\delta_K(i,j))$ where g= $e^{-(\frac{x}{B})^2}$ for every one of those obtained δ matrices of every pixel
5. Update the new pixel value using the given formula $I(i,j)^{new} = I(i,j)^{old} + \lambda[c_N(i,j)\delta_N + c_E(i,j)\delta_E + c_S(i,j)\delta_S + c_W(i,j)\delta_W]$
6. Run the algorithm several iterations for good convergence
7. Save the image

Code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 16 13:28:10 2018
```

```
@author: IIST
"""
from PIL import Image
import numpy as np
import math
import matplotlib.pyplot as plt
def g(x):
    val=(x/50)**2
    return math.exp(-val)

def get_image():
      =0.25
    img = Image.open('gaussian_noise.png').convert('L')
    img.load()
    I = np.asarray( img, dtype="uint8" )
    data=I
    print("Image is {}",data)
    N=np.zeros((I.shape[0],I.shape[1]))
    S=np.zeros((I.shape[0],I.shape[1]))
    E=np.zeros((I.shape[0],I.shape[1]))
    W=np.zeros((I.shape[0],I.shape[1]))
    cN=np.zeros((I.shape[0],I.shape[1]))
    cS=np.zeros((I.shape[0],I.shape[1]))
    cE=np.zeros((I.shape[0],I.shape[1]))
    cW=np.zeros((I.shape[0],I.shape[1]))
    I_new=np.zeros((I.shape[0],I.shape[1]))
    for k in range(20):
        for x in range(I.shape[0]-1):
            for y in range(I.shape[1]-1):
                N[x,y] = I[x-1,y] - I[x,y]
                S[x,y] = I[x+1,y] - I[x,y]
                E[x,y] = I[x,y+1] - I[x,y]
                W[x,y] = I[x,y-1] - I[x,y]

                cN[x,y] = g(N[x,y])
                cS[x,y] = g(S[x,y])
                cE[x,y] = g(E[x,y])
                cW[x,y] = g(W[x,y])

        for i in range(I.shape[0]):
            for j in range(I.shape[1]):
                sum=cN[i, j]*N[i, j] + cS[i, j]*S[i, j]+ cE[i, j]*E[i, j] + cW[i, j]*W[i
                I_new[i, j] = I[i, j] +    * sum
        I=np.copy(I_new)
    plt.imsave('original_image.png',I_new,cmap='gray')


get_image();
```

Result

Figure 1: Original Cameraman Image

Figure 2: Gaussian Noise

Figure 3: After Anisotropic diffusion

Inference

Anisotropic Diffusion was applied on the image. The difference is very much evident from smoothing filters like average/gaussian where the image is only blurred as a whole.

2. Edge Adaptive Smoothing

Add noise to your image and do edge adaptive smoothing.
Refer the book Image Processing by Petrou, page 337 Steps:

1. Read the image
2. Add gaussian noise
3. For each pixel , consider its 3x3 neighbourhood and keeping each of them as centre pixel, define 9 such kernels
4. Calculate the mean and variance of each of these 9 kernels
5. Choose the kernel with the least variance
6. Find the average of that kernel and replace the original center pixel with this average

Aim

To apply edge adaptive filtering on gaussian noised image so that edges are preserved while performing the smoothing of the image

Theory/Discussion

When an image is smoothed, a window is placed around a pixel and average value is computed inside the window and assigned to the central pixel. If the window used happens to span two different regions, the boundary between the two regions will be blurred.
In edge preserving smoothing, several windows are placed around the pixel, having the pixel in all possible relative positions with respect to the window centre. Inside each window the variance of the pixel values is computed. The window with the minimum variance is selected. The average (weighted or not) of the pixels inside that window is found and assigned to the pixel under consideration.

Algorithm

---

1. Read an Image
2. For every pixel in the image, form 9 kernels with each of its neighbors as the center point
3. Out of the kernels, choose the one with the least variance
4. Find the average value of the chosen kernel and replace the pixel value with this value
5. Repeat for every pixel
6. Save the image

Code

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 16 16:50:38 2018

@author: IIST
"""
```

```
from PIL import Image
import numpy as np
import math
import matplotlib.pyplot as plt
def get_image():
    img = Image.open('gaussian_noise.png').convert('L')
    img.load()
    I = np.asarray( img, dtype="uint8" )
    data=I
    print("Image is {}",data)
    I=np.lib.pad(I, 2, 'mean')
    I_new=np.zeros((data.shape[0],data.shape[1]))
    mean_arr=[]
    var_arr=[]
    for x in range(1,data.shape[0]):
        for y in range(1,data.shape[1]):
            one = np.matrix([[I[x-2,y-2], I[x-1,y-2],I[x,y-2]],[I[x-2,y-1], I[x-1,y-1],I
            mean_arr.append(np.mean(one))
            var_arr.append(np.var(one))
            two = np.matrix([[I[x-1,y-2], I[x,y-2],I[x+1,y-2]],[I[x-1,y-1], I[x,y-1],I[x
            mean_arr.append(np.mean(two))
            var_arr.append(np.var(two))
            thr = np.matrix([[I[x,y-2], I[x+1,y-2],I[x+2,y-2]],[I[x,y-1], I[x+1,y-1],I[x
            mean_arr.append(np.mean(thr))
            var_arr.append(np.var(thr))
            four =np.matrix([[I[x-2,y-1], I[x-1,y-1],I[x,y-1]],[I[x-2,y], I[x-1,y],I[x,y
            mean_arr.append(np.mean(four))
            var_arr.append(np.var(four))
            fiv = np.matrix([[I[x-1,y-1], I[x,y-1],I[x+1,y-1]],[I[x-1,y], I[x,y],I[x+1,y
            mean_arr.append(np.mean(fiv))
            var_arr.append(np.var(fiv))
            six = np.matrix([[I[x,y-1], I[x+1,y-1],I[x+2,y-1]],[I[x,y], I[x+1,y],I[x+2,y
            mean_arr.append(np.mean(six))
            var_arr.append(np.var(six))
            sev = np.matrix([[I[x-2,y], I[x-1,y],I[x,y]],[I[x-2,y+1], I[x-1,y+1],I[x,y+
            mean_arr.append(np.mean(sev))
            var_arr.append(np.var(sev))
            eig = np.matrix([[I[x-1,y], I[x,y],I[x+1,y]],[I[x-1,y+1], I[x,y+1],I[x+1,y+
            mean_arr.append(np.mean(eig))
            var_arr.append(np.var(eig))
            nin = np.matrix([[I[x,y], I[x+1,y],I[x+2,y]],[I[x,y+1], I[x+1,y+1],I[x+2,y+
            mean_arr.append(np.mean(nin))
            var_arr.append(np.var(nin))
            meanarr=np.array(mean_arr)
            vararr=np.array(var_arr)
            min_index=np.argmin(vararr)
            I_new[x-1,y-1]=meanarr[min_index]
            print(x,y)
            mean_arr=[]
            var_arr=[]
    plt.imsave('after_edge_smoothening.png',I_new,cmap='gray')


get_image();
```

Result

Figure 4: Original Cameraman Image

Figure 5: Gaussian Noise

Figure 6: After Edge Adaptive Smoothing

Inference

When average or Gaussian Filter is used as such to remove the noise, the edges also get blurred along with the noise. Edge Adaptive Smoothing removes additive noise while preserving the edges.