Project: Image Denoising and Edge Detection using Anisotropic Diffusion

1 Objective

The goal of this project is to implement an anisotropic diffusion model for image denoising and edge detection based on the Perona-Malik algorithm. The project will focus on developing a Python application that takes a noisy image as input and outputs a denoised image while preserving important edges.

2 Project Outline

2.1 Introduction

- Briefly explain the problem of image denoising and the importance of preserving edges.
- Introduce the Perona-Malik model and its significance in anisotropic diffusion.

2.2 Theory

- Describe the concept of scale-space filtering and its limitations.
- Explain the Perona-Malik anisotropic diffusion equation:

  $$\frac{\partial I}{\partial t} = \nabla \cdot (c(x, y, t) \nabla I)$$

  where I is the image intensity, t is the time parameter, and $c(x, y, t)$ is the diffusion coefficient.

- Discuss the choice of the diffusion coefficient:

  $$c(x, y, t) = g(|\nabla I|)$$

  where g is a monotonically decreasing function, often chosen as:

  $$g(s) = \frac{1}{1 + \left(\frac{s}{K}\right)^2}$$

or

$$g(s) = \exp\left(-\left(\frac{s}{K}\right)^2\right)$$

with K being a contrast parameter.

2.3 Implementation

2.3.1 Environment Setup

List required libraries (NumPy, SciPy, Matplotlib, PIL, etc.)

2.3.2 Loading and Preprocessing Image

```
from PIL import Image
import numpy as np

def load_image(path, gray_scale=True):
    image = Image.open(path)
    if gray_scale:
        image = image.convert('L')
    return np.array(image, dtype=np.float32)
```

2.3.3 Anisotropic Diffusion Function

```
def anisotropic_diffusion(img, num_iter, delta_t, kappa, option=1):
    h, w = img.shape
    diff_img = img.copy()
    for t in range(num_iter):
        grad_n = np.zeros_like(diff_img)
        grad_s = np.zeros_like(diff_img)
        grad_e = np.zeros_like(diff_img)
        grad_w = np.zeros_like(diff_img)

        grad_n[:-1, :] = diff_img[1:, :] - diff_img[:-1, :]
        grad_s[1:, :] = diff_img[:-1, :] - diff_img[1:, :]
        grad_e[:, :-1] = diff_img[:, 1:] - diff_img[:, :-1]
        grad_w[:, 1:] = diff_img[:, :-1] - diff_img[:, 1:]

        if option == 1:
            c_n = np.exp(-(grad_n / kappa) ** 2)
            c_s = np.exp(-(grad_s / kappa) ** 2)
            c_e = np.exp(-(grad_e / kappa) ** 2)
            c_w = np.exp(-(grad_w / kappa) ** 2)
        elif option == 2:
            c_n = 1 / (1 + (grad_n / kappa) ** 2)
            c_s = 1 / (1 + (grad_s / kappa) ** 2)
            c_e = 1 / (1 + (grad_e / kappa) ** 2)
            c_w = 1 / (1 + (grad_w / kappa) ** 2)

        diff_img += delta_t * (c_n * grad_n + c_s * grad_s + c_e * grad_e + c_w

    return diff_img
```

2.3.4 Main Script

```
import matplotlib.pyplot as plt

if __name__ == "__main__":
    image_path = 'path_to_your_image.jpg'
    img = load_image(image_path)

    num_iter = 20
    delta_t = 1/7
    kappa = 30
    option = 1

    denoised_img = anisotropic_diffusion(img, num_iter, delta_t, kappa, option)

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title("Original Image")
    plt.imshow(img, cmap='gray')
    plt.subplot(1, 2, 2)
    plt.title("Denoised Image")
    plt.imshow(denoised_img, cmap='gray')
    plt.show()
```

2.4 Results

- Compare the denoised images with the original noisy images.
- Highlight the preservation of edges and the reduction of noise.

2.5 Conclusion

- Summarize the effectiveness of the Perona-Malik anisotropic diffusion model for image denoising and edge detection.
- Discuss potential improvements and future work.

2.6 References

- Cite the Perona-Malik paper and any other resources used.

Perona-Malik Anisotropic Diffusion Algorithm

Initialization

$$I_0 = \text{input image}$$

Iterative Diffusion Process

For $t = 0$ to $t = \text{num\_iter}$:

Compute Nearest-Neighbor Differences

$$\begin{aligned}
\nabla_N I &= I_{i+1,j} - I_{i,j} \\
\nabla_S I &= I_{i-1,j} - I_{i,j} \\
\nabla_E I &= I_{i,j+1} - I_{i,j} \\
\nabla_W I &= I_{i,j-1} - I_{i,j}
\end{aligned}$$

Compute Gradient Magnitude

$$\begin{aligned}
|\nabla I_N| &= |\nabla_N I| \\
|\nabla I_S| &= |\nabla_S I| \\
|\nabla I_E| &= |\nabla_E I| \\
|\nabla I_W| &= |\nabla_W I|
\end{aligned}$$

Compute Conduction Coefficients

For Exponential Conduction Function (Option 1)

$$c_N = \exp\left(-\left(\frac{|\nabla I_N|}{K}\right)^2\right)$$

$$c_S = \exp\left(-\left(\frac{|\nabla I_S|}{K}\right)^2\right)$$

$$c_E = \exp\left(-\left(\frac{|\nabla I_E|}{K}\right)^2\right)$$

$$c_W = \exp\left(-\left(\frac{|\nabla I_W|}{K}\right)^2\right)$$

For Reciprocal Conduction Function (Option 2)

$$c_N = \frac{1}{1 + \left(\frac{|\nabla I_N|}{K}\right)^2}$$

$$c_S = \frac{1}{1 + \left(\frac{|\nabla I_S|}{K}\right)^2}$$

$$c_E = \frac{1}{1 + \left(\frac{|\nabla I_E|}{K}\right)^2}$$

$$c_W = \frac{1}{1 + \left(\frac{|\nabla I_W|}{K}\right)^2}$$

Update the Image

$$I_{i,j}^{t+1} = I_{i,j}^{t} + \Delta t \left(c_N \nabla_N I + c_S \nabla_S I + c_E \nabla_E I + c_W \nabla_W I\right)$$