ECE 6560 Final Project

Name: Pratiksha Pai **GT login:** ppai33

1 Problem Statement

Image denoising is essential for enhancing the visual quality of digital images, which is vital for applications in fields such as medical imaging, remote sensing, and automated surveillance systems, where clarity and detail accuracy are crucial.

We use Perona-Malik diffusion technique for its ability to preserve important edge details while effectively reducing noise, making it suitable for complex image processing tasks. In this study, we apply the Perona-Malik method under various conditions including different noise types like salt and pepper and Gaussian noise, different contrast levels like normal, high, and low contrast to evaluate its performance and robustness. Through these experiments, we aim to demonstrate the adaptability of the Perona-Malik technique across diverse scenarios and its potential to maintain integrity and enhance image usability in real-world applications.

2 Mathematical Formulation

Consider an image $I(x, y, t)$, where x and y denote spatial coordinates and t represents time. The goal is to reduce noise, particularly high-frequency types like salt-and-pepper or Gaussian noise. This is approached by defining an energy function that quantifies the image's gradient:

$$E(z) = \int C(z)\, dz, \tag{1}$$

where $z = \|\nabla I\|$ is the gradient magnitude.

Anisotropic diffusion, specifically the Perona-Malik method, is used to smooth the image while preserving edges by modulating the diffusion process according to the gradient's magnitude. This approach employs a variable diffusion coefficient F, defined to decrease with increasing gradient magnitude, which minimizes diffusion at edges:

$$I_t = \nabla \cdot (F(\|\nabla I\|)\nabla I), \tag{2}$$

Two common forms of the diffusion function F are:

$$F(\|\nabla I\|) = \frac{1}{1 + \left(\frac{\|\nabla I\|}{K}\right)^2}, \tag{3}$$

and

$$F(\|\nabla I\|) = e^{-\left(\frac{\|\nabla I\|}{K}\right)^2}, \tag{4}$$

where K is a parameter that adjusts the diffusion sensitivity. These functions ensure effective noise reduction while maintaining critical image features like edges.

3 Discretization & Stability Analysis

3.1 Discretization

The partial differential equation (PDE) governing the Perona-Malik diffusion, given by

$$I_t = \nabla \cdot (F(\|\nabla I\|)\nabla I),$$

is discretized for computational implementation. Here, $F = F(\|\nabla I\|)$ simplifies our expressions:

$$I_t = \nabla F \cdot \nabla I + F\Delta I \tag{5}$$
$$= F_x I_x + F_y I_y + F(I_{xx} + I_{yy}). \tag{6}$$

Using the diffusion coefficients from equations (3) we calculate derivatives:

$$F_x I_x = \frac{\partial}{\partial x}\left(\frac{1}{1+\left(\frac{I_x^2}{K^2}\right)}\right) I_x,$$

$$F_y I_y = \frac{\partial}{\partial y}\left(\frac{1}{1+\left(\frac{I_y^2}{K^2}\right)}\right) I_y.$$

The resultant diffusion equation can be reformed as:

$$I_t = \frac{K^2(I_{xx}+I_{yy}) - 2I_x^2 I_{xx} - 2I_y^2 I_{yy} - 4I_x I_y I_{xy}}{K^2 + I_x^2 + I_y^2}.$$

Using the diffusion coefficient from equation (4), the derivatives become:

$$F_x I_x = \frac{\partial}{\partial x}\left(e^{-\frac{I_x^2+I_y^2}{K^2}}\right) I_x = e^{-\frac{I_x^2+I_y^2}{K^2}}\left(-\frac{2I_x}{K^2}(2I_x I_{xx} + 2I_x I_y I_{xy})\right),$$

$$F_y I_y = \frac{\partial}{\partial y}\left(e^{-\frac{I_x^2+I_y^2}{K^2}}\right) I_y = e^{-\frac{I_x^2+I_y^2}{K^2}}\left(-\frac{2I_y}{K^2}(2I_y I_{yy} + 2I_x I_y I_{xy})\right)$$

Combining these terms into the diffusion equation, we have:

$$I_t = \frac{e^{-\frac{I_x^2+I_y^2}{K^2}} \left(K^2(I_{xx}+I_{yy}) - 2I_x^2 I_{xx} - 2I_y^2 I_{yy} - 4I_x I_y I_{xy}\right)}{K^2}$$

This expression encapsulates the effect of the exponential diffusion coefficient, effectively controlling the diffusion near edges based on the local gradient magnitudes.

Discretization for numerical computation utilizes forward and central differences:

$$I_t = \frac{I(t+\Delta t) - I(t)}{\Delta t}, \tag{7}$$

$$I_x = \frac{I(x+\Delta x) - I(x-\Delta x)}{2\Delta x}, \tag{8}$$

$$I_y = \frac{I(y+\Delta y) - I(y-\Delta y)}{2\Delta y}, \tag{9}$$

$$I_{xx} = \frac{I(x+\Delta x) - 2I(x) + I(x-\Delta x)}{\Delta x^2}, \tag{10}$$

$$I_{yy} = \frac{I(y+\Delta y) - 2I(y) + I(y-\Delta y)}{\Delta y^2}, \tag{11}$$

$$\begin{aligned} I_{xy} = \frac{1}{4\Delta x \Delta y}\big[&I(x+\Delta x, y+\Delta y) + I(x-\Delta x, y-\Delta y) \\ &- I(x+\Delta x, y-\Delta y) - I(x-\Delta x, y+\Delta y)\big] \end{aligned} \tag{12}$$

This simplified mathematical framework allows for the effective numerical solution of the Perona-Malik diffusion equation in image processing tasks.

3.2 Stability Analysis

The discretization of the Perona-Malik equation is widely acknowledged for its stability and is utilized in this project following the well-established method in the literature. We adopt spatial steps $\Delta x = \Delta y = 1$ and define the time step Δt accordingly:

$$\begin{aligned} I(x,y,t+1) = I(x,y,t) + \Delta t\big[&C_N \nabla_N I(x,y,t) + C_S \nabla_S I(x,y,t) \\ &+ C_E \nabla_E I(x,y,t) + C_W \nabla_W I(x,y,t)\big], \end{aligned} \tag{13}$$

where the gradient components are defined as:

$$\nabla_N I(x,y,t) = I(x-1,y,t) - I(x,y,t), \tag{14}$$

$$\nabla_S I(x,y,t) = I(x+1,y,t) - I(x,y,t), \tag{15}$$

$$\nabla_E I(x,y,t) = I(x,y+1,t) - I(x,y,t), \tag{16}$$

$$\nabla_W I(x,y,t) = I(x,y-1,t) - I(x,y,t). \tag{17}$$

For stability, the time step Δt must satisfy the Courant-Friedrichs-Lewy (CFL) condition. By adopting a simplification from the one-dimensional heat equation stability analysis, where $\Delta t \leq \frac{(\Delta x)^2}{2F}$, and recognizing the symmetry in two dimensions, we determine the CFL condition for this 2D case as:

$$\Delta t \leq \frac{1}{4F}, \tag{18}$$

assuming F represents the maximum diffusivity within the image and is bounded between 0 and 1. Consequently, a $\Delta t \leq 0.25$ ensures the scheme's convergence, accommodating the typical bounds of F from 0 to 1.

4 Error Calculation

Metrics for assessing denoised image quality include:

4.1 Peak Signal-to-Noise Ratio (PSNR)

PSNR measures the ratio of the maximum possible pixel intensity of the image to the power of the distortion noise. It is defined as:

$$\text{PSNR}(\hat{v}, v) = 20 \log_{10} \left(\frac{\max(v)}{\sqrt{\text{MSE}(\hat{v}, v)}} \right),$$

where $\hat{v}$ is the denoised image, v the original image, and $\max(v)$ the maximum pixel value in v. Higher PSNR values indicate superior denoising quality.

4.2 Mean Square Error (MSE)

MSE quantifies the average squared intensity differences between the denoised and original images:

$$\text{MSE}(\hat{v}, v) = \frac{1}{n} \sum_{i=1}^{n} (\hat{v}_i - v_i)^2,$$

with lower MSE values signifying better denoising performance.

5 Experiments

5.1 Sensitivity of Diffusion Coefficient to the Gradient

An experiment was conducted to analyze the sensitivity of the diffusion coefficient F to changes in the image gradient for different values of the parameter K. For this purpose, a range of image gradients was simulated, and the corresponding diffusion coefficient values were computed using the diffusion functions F_1 and F_2 as defined in equations (3) and (4), respectively.

Below plots show the sensitivity of the diffusion coefficient to the gradient.

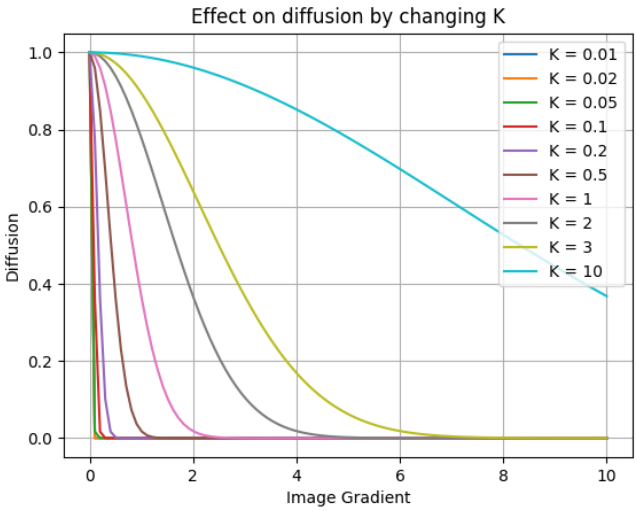

Figure 1: The sensitivity of the diffusion coefficient F_1 to the image gradient for different values of K.

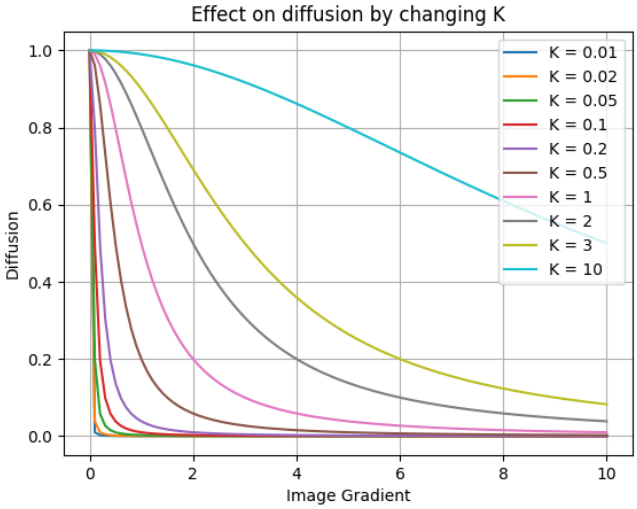

Figure 2: The sensitivity of the diffusion coefficient F_2 to the image gradient for different values of K.

5.2 PSNR Sensitivity Analysis

Experiments were conducted to evaluate the sensitivity of PSNR to the Perona-Malik denoising parameters: diffusion constant K, time step Δt, and iteration count. The diffusion functions F_1 and F_2, associated with Gaussian and salt-and-pepper noise, were analyzed. Plots correlating PSNR with each parameter helped optimise the settings for maximum denoising efficiency.

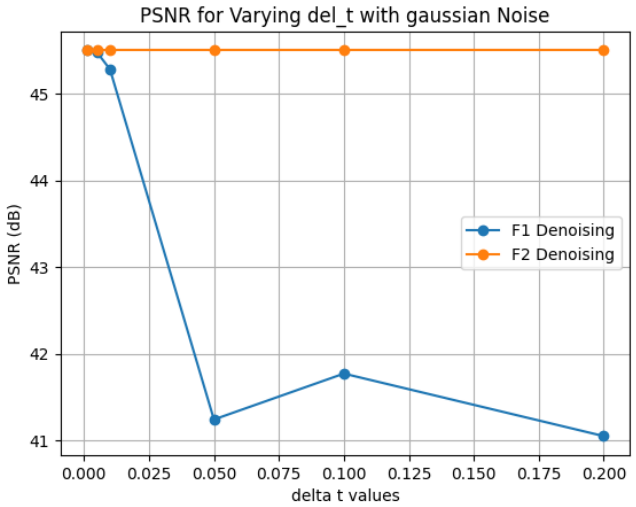

Figure 3: PSNR vs Δt for Gaussian Noise.

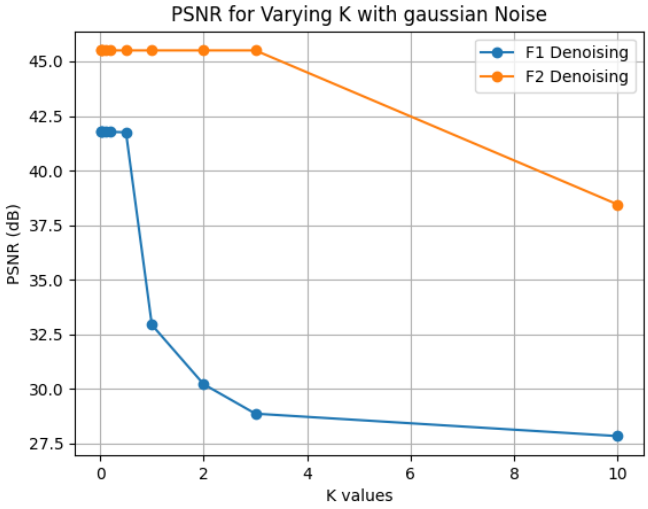

Figure 4: PSNR vs K for Gaussian Noise.

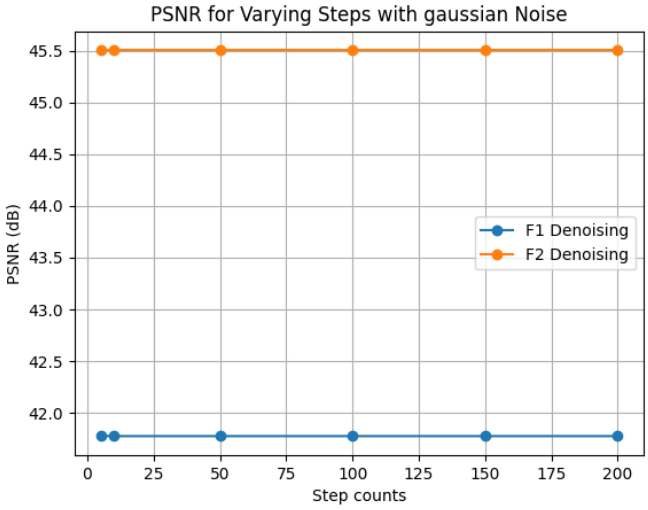

Figure 5: PSNR vs Steps for Gaussian Noise.

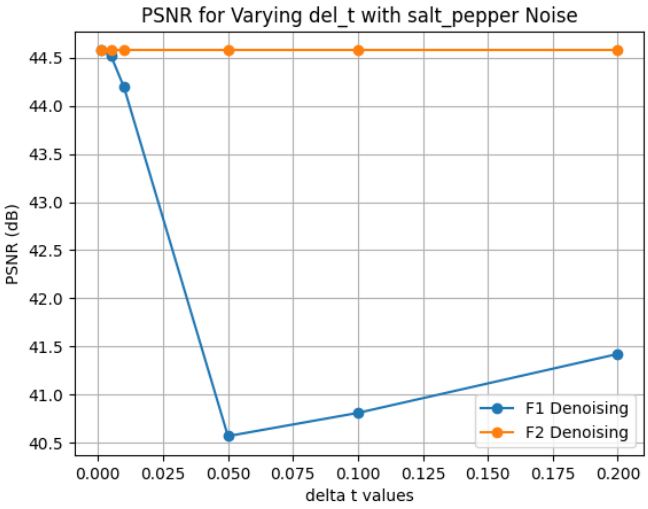

Figure 6: PSNR vs Δt for Gaussian Noise.

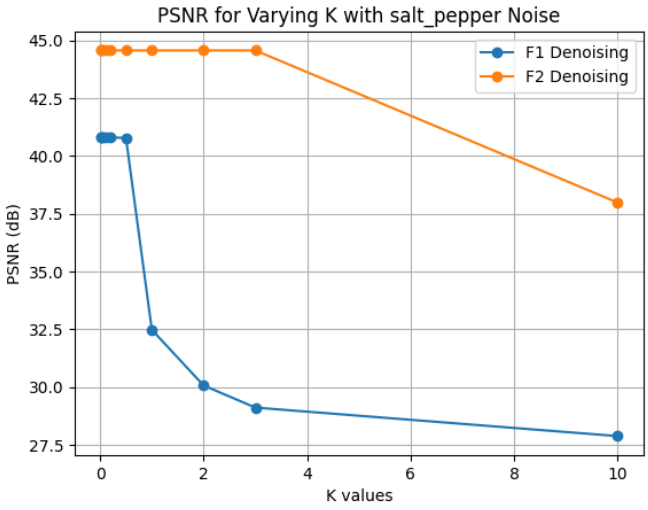

Figure 7: PSNR vs K for Gaussian Noise.

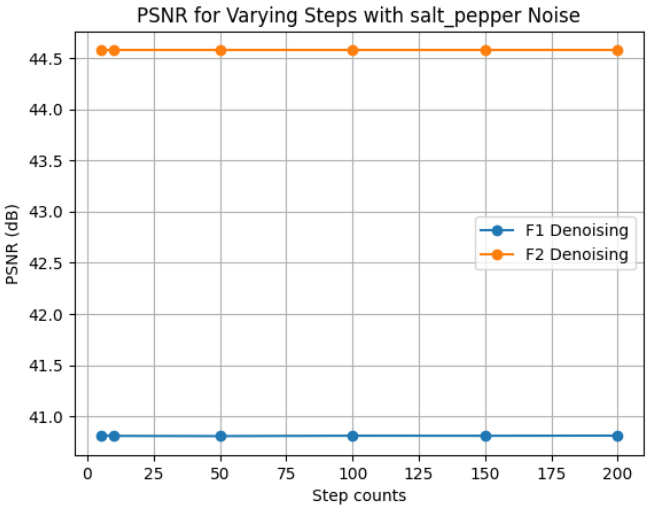

Figure 8: PSNR vs Steps for Gaussian Noise.

5.3 Denoising Performance Under Varying Contrast Conditions

A comprehensive evaluation was conducted to assess the effectiveness of different denoising techniques under various contrast conditions: normal, high contrast, and low contrast. Each condition was tested against two prevalent noise types: Gaussian and salt-and-pepper. The denoising methods examined include Gaussian filtering, wavelet denoising, and two variants of Perona-Malik diffusion (F1 and F2). Performance metrics used in this analysis are Peak Signal-to-Noise Ratio (PSNR) and Mean Squared Error (MSE).

In order to get the optimal parameters for Perona-Malik diffusion methods a grid search was employed on all different noisy images. The parameters included the diffusion constant K, the time step Δt, and the number of iterations (steps). The optimization aimed to maximize the PSNR and minimize the MSE relative to the original image.

Normal Contrast Conditions As previously detailed, the denoising methods were first applied to images with normal contrast levels. The results of this are shown in the following table and figures:

Table:

Table 1: Optimal Parameters and Corresponding PSNR and MSE Values for Original Denoised Images

Noise Type	Denoising Method	Optimal K	Optimal Steps	PSNR (dB) / MSE
Gaussian	Wavelet Denoising	N/A	N/A	25.28 / 192.87
Gaussian	Gaussian Denoising	N/A	N/A	33.15 / 31.47
Gaussian	Perona-Malik F1	0.01	5	44.25 / 2.44
Gaussian	Perona-Malik F2	0.01	5	44.56 / 2.28
Salt & Pepper	Wavelet Denoising	N/A	N/A	31.69 / 44.05
Salt & Pepper	Gaussian Denoising	N/A	N/A	34.02 / 25.79
Salt & Pepper	Perona-Malik F1	0.01	5	45.30 / 1.92
Salt & Pepper	Perona-Malik F2	0.01	5	45.51 / 1.83

Figures:

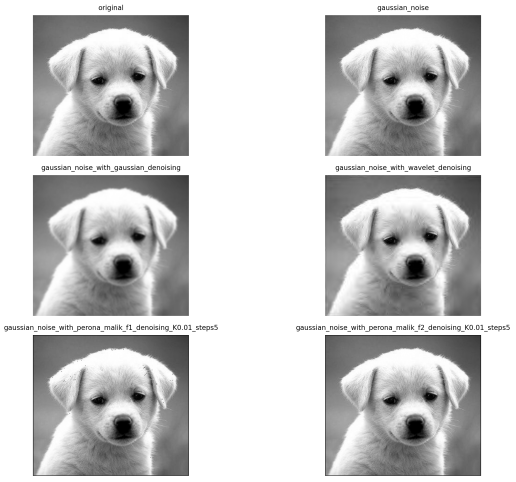

Figure 9: Gauss Noise Denoising process

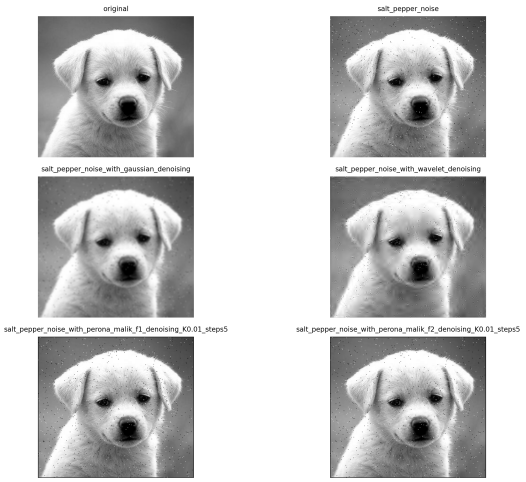

Figure 10: Salt and Pepper Noise Denoising process

High Contrast Conditions Subsequently, the same denoising methods were applied to images enhanced with high contrast to evaluate their robustness and efficiency in more challenging scenarios. The optimal parameters and their corresponding performance metrics are documented in:

Table:

Table 2: Optimal Parameters and Corresponding PSNR and MSE Values for High Contrast Images

Noise Type	Denoising Method	Optimal K	Optimal Steps	PSNR (dB) / MSE
Salt & Pepper	Wavelet Denoising	N/A	N/A	20.72 / 550.63
Salt & Pepper	Gaussian Denoising	N/A	N/A	28.65 / 88.68
Salt & Pepper	Perona-Malik F1	10	5	28.83 / 85.09
Salt & Pepper	Perona-Malik F2	0.01	5	28.67 / 88.23
Gaussian	Wavelet Denoising	N/A	N/A	21.78 / 431.60
Gaussian	Gaussian Denoising	N/A	N/A	28.65 / 88.83
Gaussian	Perona-Malik F1	0.5	5	28.66 / 88.58
Gaussian	Perona-Malik F2	0.01	5	28.65 / 88.63

Figures:

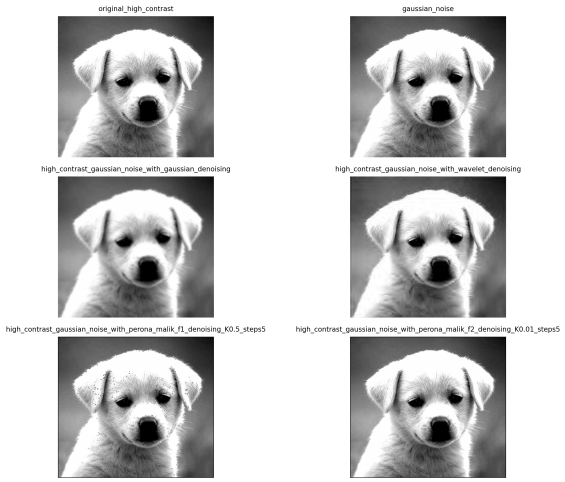

Figure 11: Gauss Noise Denoising process

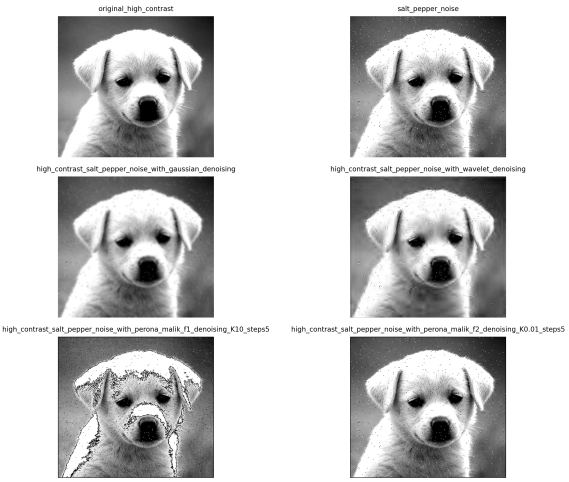

Figure 12: Salt and Pepper Noise Denoising process

Low Contrast Conditions Lastly, the denoising techniques were tested on images with low contrast. This condition typically poses a challenge as the noise can be more pronounced relative to the actual image content. The performance metrics obtained are as follows:

Table:

Table 3: Optimal Parameters and Corresponding PSNR and MSE Values for Low Contrast Images

Noise Type	Denoising Method	Optimal K	Optimal Steps	PSNR (dB) / MSE
Salt & Pepper	Wavelet Denoising	N/A	N/A	18.05 / 1018.90
Salt & Pepper	Gaussian Denoising	N/A	N/A	28.01 / 102.90
Salt & Pepper	Perona-Malik F1	1	5	28.01 / 102.72
Salt & Pepper	Perona-Malik F2	0.01	5	27.99 / 103.25
Gaussian	Wavelet Denoising	N/A	N/A	18.83 / 851.17
Gaussian	Gaussian Denoising	N/A	N/A	28.09 / 100.85
Gaussian	Perona-Malik F1	1	5	28.19 / 98.65
Gaussian	Perona-Malik F2	0.01	5	28.12 / 100.23

Figures:

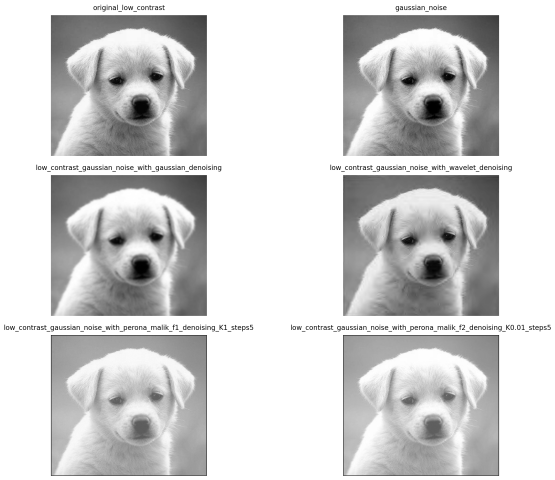

Figure 13: Gauss Noise Denoising process

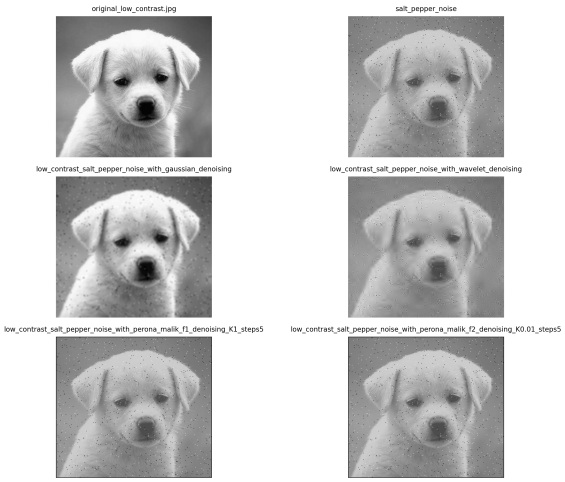

Figure 14: Salt and Pepper Noise Denoising process

Comparative Analysis The experiments showcase different denoising method across different contrast levels and noise types. Notably, Perona-Malik diffusion methods often achieve higher PSNR values, particularly in high contrast scenarios, demonstrating its robustness. These results underline the importance of choosing an appropriate denoising technique based on the specific characteristics of the image data and the noise present.

6 Discussion and Future Work

The evaluation demonstrated the robustness of Perona-Malik methods, particularly for different noisy images, emphasizing its usage for detail-sensitive applications. Contrastingly, certain methods like wavelet denoising performed badly under specific noise conditions. These findings underscore the importance of method selection tailored to image characteristics and application demands.

Future research directions include optimizing the Perona-Malik algorithm for dynamic noise profiles, leveraging deep learning for data-driven denoising, accelerating algorithms for real-time application, and extensive validation against diverse real-world datasets.

7 Code structure

The code repository, accessible at `https://github.com/pratikshapi/ece6560-pm`, is structured into directories corresponding to different stages of the image processing workflow. Core functionalities are encapsulated within Python modules: `noise.py` handles the generation of noisy images; `denoise.py` implements various denoising algorithms, including the Perona-Malik diffusion; `metrics.py`

computes PSNR and MSE; and `util.py` provides utilities for directory management and image validation. The main script, `main.py`, orchestrates the denoising process, calling upon the different modules to perform tasks like grid search, plotting results, and saving denoised images. There are some additional scripts are provided to process images with variable parameters and generate plots for the diffusion coefficients.

Some code snippets can be found below:

```python
def main():
    logger.info("Starting denoising process from main.py...")
    original_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    validate_image(original_image)
    base_directory = "denoised/"
    noise_funcs = [add_salt_pepper_noise, add_gaussian_noise]
    noise_labels = ['salt_pepper_noise', 'gaussian_noise']
    generate_noised_images_if_absent(image_path, noise_funcs, noise_labels, ['normal'])
    denoise_funcs = [wavelet_denoising, gaussian_denoise, anisodiff_f1, anisodiff_f2]
    denoise_labels = ['wavelet_denoising', 'gaussian_denoising', 'perona_malik_f1_denoising', 'perona_malik_f2_denoising']

    noisy_images = generate_noisy_images(original_image, noise_funcs)
    process_and_save_images(noisy_images, noise_labels, denoise_funcs, denoise_labels, base_directory, K_values, step_counts)
```

Figure 15: main.py - main function

```python
def process_and_save_images(noisy_images, noise_labels, denoise_funcs, denoise_labels, base_directory="", K_values=None, step_counts=None):
    logger = setup_logger()  # Ensure the logger is set up once here

    for noisy, noise_label in zip(noisy_images, noise_labels):
        directory = f"{base_directory}{noise_label}/"

        for denoise_func, denoise_label in zip(denoise_funcs, denoise_labels):
            if 'perona_malik' in denoise_label:
                # Use the optimization function to get the best parameters and images
                best_K, best_steps, best_psnr, best_mse = optimize_pmf_parameters(noisy.copy(), K_values, step_counts, denoise_func)
                # Generate the best denoised image using the optimal parameters
                best_image = denoise_func(noisy.copy(), steps=best_steps, K=best_K)
                # Save the best denoised image
                denoised_filename = f'{directory}{noise_label}_with_{denoise_label}_K{best_K}_steps{best_steps}.png'
                save_image(denoised_filename, best_image)
                # Log the best PSNR and MSE values
                logger.info(f"{denoise_label} with K={best_K} and steps={best_steps}: Best PSNR = {best_psnr}, Best MSE = {best_mse}")
            else:
                # Process non-Perona-Malik methods directly
                denoised_image = denoise_func(noisy.copy())
                denoised_filename = f'{directory}{noise_label}_with_{denoise_label}.png'
                save_image(denoised_filename, denoised_image)
                # Calculate and log PSNR and MSE for direct methods
                psnr_value = psnr(original_image, denoised_image)
                mse_value = mse(original_image, denoised_image)
                logger.info(f"{denoise_label}: PSNR = {psnr_value}, MSE = {mse_value}")
```

Figure 16: main.py - process_and_save_images

```python
def F1(dt, K):
    return np.exp(-1 * (np.power(dt, 2)) / (np.power(K, 2)))

def F2(dt, K):
    func = 1 / (1 + ((dt/K)**2))
    return func

def anisodiff_f1(image, steps=10, K=0.1, del_t=0.01, debug=False):
    denoised_image = np.zeros(image.shape, dtype=image.dtype)
    for i in range(steps):
        north_diff = image[:-2, 1:-1] - image[1:-1, 1:-1]
        south_diff = image[2:, 1:-1] - image[1:-1, 1:-1]
        east_diff = image[1:-1, 2:] - image[1:-1, 1:-1]
        west_diff = image[1:-1, :-2] - image[1:-1, 1:-1]
        denoised_image[1:-1, 1:-1] = image[1:-1, 1:-1] + del_t * (
            F1(north_diff, K) * north_diff + F1(south_diff, K) * south_diff +
            F1(east_diff, K) * east_diff + F1(west_diff, K) * west_diff)
        image = denoised_image.copy()

        # Visual debugging
        if debug and i%10 == 0:
            plt.figure(figsize=(4, 4))
            plt.imshow(image, cmap='gray')
            plt.title(f'Step {i+1}')
            plt.axis('off')
            plt.show()

    return image

def anisodiff_f2(image, steps=50, K=4, del_t=0.01, debug=False):
    denoised_image = np.zeros(image.shape, dtype=image.dtype)
    for i in range(steps):
        north_diff = image[:-2, 1:-1] - image[1:-1, 1:-1]
        south_diff = image[2:, 1:-1] - image[1:-1, 1:-1]
        east_diff = image[1:-1, 2:] - image[1:-1, 1:-1]
        west_diff = image[1:-1, :-2] - image[1:-1, 1:-1]
        denoised_image[1:-1, 1:-1] = image[1:-1, 1:-1] + del_t * (
            F2(north_diff, K) * north_diff + F2(south_diff, K) * south_diff +
            F2(east_diff, K) * east_diff + F2(west_diff, K) * west_diff)
        image = denoised_image.copy()

        # Visual debugging
        if debug and i%10 == 0:
            plt.figure(figsize=(4, 4))
            plt.imshow(image, cmap='gray')
            plt.title(f'Step {i+1}')
            plt.axis('off')
            plt.show()

    return image
```

Figure 17: denoise.py - Perona Malik functions