Guide for Running the Code for Medical Image Denoising

To execute this code and achieve optimal results, here's a comprehensive step-by-step guide outlining the prerequisites, detailed execution process, and key considerations. This guide takes you through each stage of the code, from preprocessing images to final evaluation using PSNR and SSIM metrics. Essential software packages and tools required for proper code execution are also listed.

Prerequisites and Requirements

To run this code effectively in MATLAB, you'll need to install certain toolboxes and packages. These tools are essential for leveraging CNN and autoencoder models to denoise medical images. Check and install the following items before proceeding:

1. **MATLAB R2021a** or later.
2. **Required Toolboxes**:
   - *Image Processing Toolbox*: Useful for initial image processing, filtering, and various image operations.
   - *Deep Learning Toolbox*: Required for implementing CNNs and autoencoders. This toolbox includes functions for designing, training, and testing deep networks.
   - *Computer Vision Toolbox*: Needed for some preprocessing and postprocessing functions.

Execution Steps and Explanations

Below are the steps broken down in detail to help you understand the function of each section.

Step 1: Data Loading and Preparation

First, load both the original (clean) image and the reference image. Images should be in RGB format, and if they are grayscale, they should be converted to RGB. Then, normalize the images to decimal values.

Step 2: Adding Noise to the Image

In this step, **Rayleigh noise** is added to the original image. The noise level is set to 6% of the maximum intensity to simulate realistic image noise. This step generates Rayleigh noise by calculating two random noise elements and combining them.

Step 3: Applying the Wiener Filter

The Wiener filter is applied to each channel of the noisy image to reduce initial noise, helping to enhance the image quality before deeper processing.

Step 4: Setting Fast Bilateral Filter Parameters

Bilateral filter parameters are automatically adjusted based on the intensity of the reference image. The fast bilateral filter is applied to further reduce noise while preserving image details.

Step 5: Applying Unsharp Masking for Image Sharpness

To enhance the image sharpness, the *Unsharp Mask* filter is applied. This filter is adjusted with parameters like radius and amount to improve detail clarity.

Step 6: Designing a Convolutional Neural Network (CNN) with Residual Connections

This section defines the CNN architecture. The network consists of convolutional layers with 64 filters and incorporates residual connections to allow information flow from previous layers to subsequent ones, which improves the network's performance in reconstructing images.

Step 7: Setting CNN Training Parameters

Training parameters, including learning rate, number of epochs, and mini-batch size, are set in this section. The **Adam** optimizer is used to optimize the network parameters.

Step 8: Training the CNN

The CNN is trained using the preprocessed image (filtered by earlier stages) as input and the clean image as the target. The training progress is visualized in a plot.

Step 9: Evaluating and Predicting CNN Output

The trained CNN model is applied to the input image to generate the denoised output, which is then passed to the next processing stage.

Step 10: Designing the Autoencoder Architecture

The autoencoder, comprising convolutional and transposed convolutional layers, is designed to help reconstruct image details. This model addresses image blurriness and enhances overall sharpness.

Step 11: Setting Autoencoder Training Parameters

Autoencoder training parameters—similar to those of the CNN—are configured here, including learning rate, epochs, and mini-batch size.

Step 12: Training the Autoencoder

The autoencoder is trained using the CNN output as input and the clean image as the target. The training progress is displayed with a plot.

Step 13: Evaluating and Predicting Autoencoder Output

The autoencoder is applied to reconstruct the final image output. It improves image quality by further reducing remaining noise.

Step 14: Applying Unsharp Masking to the Final Image

The *Unsharp Mask* is applied again to the final output image to optimize sharpness and enhance detail clarity.

Step 15: Calculating PSNR and SSIM Metrics for Final Evaluation

Lastly, **PSNR** and **SSIM** metrics are calculated to evaluate the quality of the final image. These metrics indicate the similarity of the output to the clean image and its overall quality.

Step 16: Plotting PSNR and SSIM Values

A plot is generated to visually compare the model's performance based on PSNR and SSIM values.

Plotting Weights and Biases of CNN and Autoencoder Layers

The weights and biases of the convolutional layers in both the CNN and autoencoder are extracted and plotted to provide a visual overview of their distribution and initialization.

Final Notes

- **Training Parameters**: Fine-tuning parameters like learning rate and number of filters can significantly impact the final results.
- **Potential Improvements**: Adding more complex layers, such as Group Normalization layers, may further enhance the network’s performance.

By following these steps, you will likely obtain a high-quality, denoised image. This result leverages an optimized model combining CNN and autoencoder approaches, enhanced by pre- and post-processing techniques.