1) Network Design: I opted for a shallow autoencoder to keep the architecture simple and avoid unnecessary complexity. Autoencoders are effective for denoising tasks as they map the image domain to a latent space and then reconstruct the images from that latent space. The choice of a Convolutional Autoencoder was motivated by the powerful capabilities of CNNs in handling image data. While autoencoders can be implemented with fully connected networks and image flattening, the use of CNNs allows direct manipulation of the image's spatial structure. ReLU activation functions were used, except for the last layer, due to the data being normalized within [0, 1], making sigmoid output more suitable for this range. As regularization batch normalization is used.

2) Training: For training, the Mnist10 dataset was used as asked. The dataset was preprocessed in a separate dataloader.py file to improve code readability.

3) Adding Noise: Gaussian noise was added to the images. Since the images were normalized between [0, 1], noisy pixel values were clamped to this range to prevent out-of-bound values. The noise characteristic can be varied during training to achieve more general-purpose networks, but the main purpose of denoising autoencoders remains intact. This approach enhances the autoencoder's robustness to different noise patterns.

4) Model Testing and Saving: PyTorch's built-in model saving strategies were employed, saving the model every 10 epochs. Testing was performed on different images than those used for training.

5) Loss Selection and Comparison Criteria: Mean Squared Error (MSE) was chosen as the loss function, which is well-suited for optimizing denoising tasks. To compare the denoising performance of the autoencoder with that of wavelet denoising, the Peak Signal-to-Noise Ratio (PSNR) was used. PSNR is a common metric for evaluating image compression, denoising, and reconstruction performance. If a different one is necessary MSSIM can be used.

6) Output: The proposed design yields the following outputs:

- Original Image
- Noisy Image
- Wavelet Denoised Image
- Autoencoder Denoised Image

7) Code Testing: Python unittest was used to create a test script, ensuring the reliability and correctness of the code. The main properties tested include:

- Dataloader output to verify correct shape.
- Pixel value range of loaded images, ensuring the model is designed for the input range [0, 1].
- Network output shape, confirming that the denoised images have the same resolution as the input images.

8) Documentation: A ReadMe file is prepared to make more understandable and requirements.txt is introduced.

9) Problems: During the course of this project, I encountered no major challenges. However, there are potential areas for further development that could enhance the performance of the autoencoder network. For instance, I could explore the use of different noise models, inspired by diffusion models, where a mixture of Gaussians is applied as noise at each epoch. Additionally, employing a deeper network architecture with skip connections to preserve information or residual connections to preserve gradients might lead to improved results. Furthermore, experimenting with different optimizers and hyperparameters could optimize the model's training process.

Despite the potential for these improvements, I chose not to implement them in this iteration of the project. My belief is that maintaining the clarity and cohesiveness of the underlying logic is of paramount importance. To evaluate the performance of the trained network, I conducted tests using diverse inputs from various datasets, including Clic and Kodak. Notably, the proposed autoencoder network outperformed a wavelet denoiser using the Bayes method, which itself is superior to the visushrink method.

10) Time Period: The total time invested in this project was approximately 8-10 hours. Within that timeframe, I achieved results within 4-6 hours on average. Throughout the process, I explored different approaches, such as model redesign by making it shallower or training without batch normalization. Additionally, I made efforts to enhance code readability by organizing each part into separate functions.

Thanks for the task, it was fun ☺.

Ugur Berk Sahin