Department of electrical engineering

Medical image processing course

Dr. Fatemizadeh

The third exercise

Zahra Sorkhai

98101725

Theory Part

Question1:

The zero norm counts how many elements of x are non-zero. For each element x_i, the expression is zero if x_i is zero, because 1-1=0. Otherwise, if x_i is not zero, then x_i^2 is positive and -beta is negative, so the exponential term goes to zero as beta goes to negative infinity. This makes the expression equal to one for each non-zero x_i. Therefore, the sum of the expression over all x_i gives the number of non-zero elements.

Question2:

CBDNet:

It consists of two subnetworks: a noise estimation subnetwork and a non-blind denoising subnetwork. The noise estimation subnetwork takes a noisy image as input and outputs a noise map that estimates the noise level at each pixel. The non-blind denoising subnetwork takes the noisy image and the noise map as inputs and outputs a clean image. The network is trained on both synthetic images based on a signal-dependent noise model and real photographs with ground-truth images. The network can handle unknown or varying noise levels or types, without requiring additional information about the noise.

The main steps of CBDNet are:

- Preprocessing: The input noisy image is converted from sRGB to linear RGB space, and then normalized to [0, 1] range.
- Noise estimation: The noise estimation subnetwork is composed of several convolutional layers with different kernel sizes and dilation rates, followed by a sigmoid activation function. The network learns to estimate the noise map from the noisy image, using asymmetric learning to penalize under-estimation more than over-estimation of noise level.
- Non-blind denoising: The non-blind denoising subnetwork is composed of several residual blocks with dilated convolutions, followed by a convolutional layer and a ReLU activation function. The network learns to remove the

estimated noise from the noisy image, using perceptual loss to preserve the details and textures of the clean image.

- Postprocessing: The output clean image is denormalized to [0, 255] range, and then converted from linear RGB to sRGB space.

GCBD:

It is a generative adversarial network (GAN) that consists of a generator and a discriminator. The generator is a convolutional neural network (CNN) that takes a noisy beamspace channel matrix as input and outputs a denoised beamspace channel matrix. The discriminator is also a CNN that takes a beamspace channel matrix as input and outputs a probability of whether it is real or fake. The network is trained on synthetic data with different noise levels and types, and can handle blind denoising without knowing the noise information.

The main steps of GCBD are:

- Preprocessing: The input noisy beamspace channel matrix is normalized to [0, 1] range, and then reshaped into an image-like tensor of size 64x64x2.
- Denoising: The generator subnetwork is composed of several convolutional layers with different kernel sizes and filters, followed by batch normalization layers and ReLU activation functions. The network learns to estimate the denoised beamspace channel matrix from the noisy one, using Wasserstein loss with gradient penalty as the objective function. The discriminator subnetwork is composed of several convolutional layers with different kernel sizes and filters, followed by LeakyReLU activation functions. The network learns to distinguish the real beamspace channel matrix from the fake one, using Wasserstein loss as the objective function.
- Postprocessing: The output denoised beamspace channel matrix is reshaped into a matrix of size 64x128, and then denormalized to the original range.

SLSR-Net:

It is a deep convolutional neural network that takes a noisy sinogram as input and outputs a clean sinogram. The network is trained on both synthetic data with

different noise levels and types, and real-world data with unknown noise. The network consists of two sub-networks: a supervised sub-network and an unsupervised sub-network. The supervised sub-network learns to restore the sinogram from paired low-dose/high-dose sinograms, while the unsupervised sub-network learns to restore the sinogram from unpaired low-dose sinograms using a weighted least-squares model with a regularization term.

The main steps of SLSR-Net are:

- Preprocessing: The input noisy sinogram is normalized to [0, 1] range, and then cropped into patches of size 64x64.
- Restoration: The supervised sub-network is composed of several convolutional layers with different kernel sizes and filters, followed by batch normalization layers and ReLU activation functions. The network learns to estimate the clean sinogram from the low-dose sinogram, using mean squared error (MSE) loss as the objective function. The unsupervised sub-network is composed of several convolutional layers with different kernel sizes and filters, followed by batch normalization layers and ReLU activation functions. The network learns to estimate the clean sinogram from the low-dose sinogram, using a weighted least-squares model with a regularization term as the objective function. The two sub-networks are connected by a Kullback-Leibler divergence loss that transfers the learned feature distribution from the supervised sub-network to the unsupervised sub-network.
- Postprocessing: The output clean sinogram is denormalized to [0, 255] range, and then reconstructed into a CT image using the filtered backprojection algorithm.

FFDNet:

It takes a noisy image and a noise level map as inputs and outputs a clean image. The noise level map can be either uniform or non-uniform, allowing the network to handle different noise levels or types. The network is trained on synthetic images with additive white Gaussian noise (AWGN) and real photographs with unknown

noise. The network can achieve a good trade-off between inference speed and denoising performance by working on downsampled sub-images.

The main steps of FFDNet are:

- Preprocessing: The input noisy image is normalized to [0, 1] range, and then divided into sub-images with overlapping borders. The sub-images are downsampled by a factor of 2 using bicubic interpolation. The noise level map is also downsampled by the same factor.
- Denoising: The denoising subnetwork is composed of several convolutional layers with different kernel sizes and dilation rates, followed by a batch normalization layer and a ReLU activation function. The network learns to remove the noise from the downsampled sub-images, using mean squared error (MSE) loss as the objective function.
- Postprocessing: The output sub-images are upsampled by a factor of 2 using bicubic interpolation, and then merged into a full-sized image by averaging the overlapping regions. The output image is denormalized to [0, 255] range.

DnCNN:

It takes a noisy image as input and outputs a residual image that represents the noise. The clean image can be obtained by subtracting the residual image from the noisy image. The network is trained on synthetic images with additive white Gaussian noise (AWGN) and can handle different noise levels with a single model. The network uses residual learning and batch normalization to speed up the training process and improve the denoising performance.

The main steps of DnCNN are:

- Preprocessing: The input noisy image is normalized to [0, 1] range, and then cropped into patches of size 40x40.

- Denoising: The denoising subnetwork is composed of 17 convolutional layers with 3x3 kernels and 64 filters, except for the first and the last layers which have 1 filter. The first layer is followed by a ReLU activation function, and the other layers are followed by a batch normalization layer and a ReLU activation function. The network learns to estimate the residual image from the noisy image, using mean squared error (MSE) loss as the objective function.
- Postprocessing: The output residual image is denormalized to [0, 255] range, and then subtracted from the input noisy image to obtain the clean image.

DnCNN and FFDNet are both deep convolutional neural networks for image denoising, but they have some differences in their architectures and inputs. Some of the main differences are:

- DnCNN takes a noisy image as input and outputs a residual image that represents the noise. FFDNet takes a noisy image and a noise level map as inputs and outputs a clean image. The noise level map can be either uniform or non-uniform, allowing FFDNet to handle spatially variant noise.
- DnCNN works on full-sized images, while FFDNet works on downsampled sub-images. This makes FFDNet faster and more memory-friendly than DnCNN.
- DnCNN uses 17 convolutional layers with 3x3 kernels and 64 filters, except for the first and the last layers which have 1 filter. FFDNet uses 12 convolutional layers with different kernel sizes and dilation rates, and 96 filters for all layers.
- DnCNN uses mean squared error (MSE) loss as the objective function, while FFDNet uses a combination of MSE loss and perceptual loss. Perceptual loss is computed by comparing the features extracted by a pre-trained VGG network from the clean image and the output image.

Preactical Part

Question 1:

TV_GPCL:

The function is called TV_GPCL, which stands for Total Variation Gradient Projection with Constant Length. It is a program that applies a basic gradient projection method with constant step length to solve the dual formulation of the ROF image restoration model. The ROF model is a technique for removing noise from images by minimizing the total variation of the image subject to a fidelity constraint.

The function takes as input variables:

- w1, w2: Dual variables, initial guess.
- f: Noisy image
- lbd: Constant fidelity parameter
- alpha: Fixed step length
- NIT: Maximum number of iterations
- GapTol: Convergence tolerance (stop criterion) for relative duality gap
- verbose: A flag to print intermediate results

The function returns as output variables:

- u: Primal variable, numerical solution - restored image
- w = (w1,w2): Dual variable, numerical solution
- Energy: The value of the objective function
- DGap: Duality Gap
- TimeCost: CPU time cost
- itr: Number of iterations

The function works as follows:

- It initializes some variables such as g, gx, gy, sf, which are related to the noisy image f and the fidelity parameter lbd.

- It computes the initial energy, which is the objective function value of the dual problem. The objective function is the squared norm of the divergence of w minus lambda times f.
- It computes the initial primal variable u and the initial duality gap. The primal variable is obtained by subtracting f from the divergence of w divided by lambda. The duality gap is obtained by summing the norm of the gradient of u and the inner product of u and w.

It starts a loop for a maximum of NIT iterations or until the relative duality gap is less than GapTol.

- It computes the gradient of the objective function with respect to w, which is given by dFx and dFy.
- It performs a gradient projection step with constant step length alpha. This means that it updates w by subtracting alpha times the gradient and then projecting it onto the feasible set, which is the unit ball. The projection is done by dividing w by its norm if it exceeds one.
- It computes the new energy value using the updated w.
- It computes the new primal variable u and the new duality gap using the updated w.
- It tests for convergence by checking if the relative duality gap is less than GapTol. If so, it exits the loop. Otherwise, it continues to the next iteration.

The gradient projection method is a method for solving bound constrained optimization problems by searching along a piecewise linear path that consists of projecting a point onto the feasible set after subtracting a multiple of the gradient. It is an efficient and simple method that allows large changes in the working set at each iteration.

TV_Chambolle:

The function is called TV_Chambolle, which stands for Total Variation Chambolle's method. It is a program that applies Chambolle's semi-implicit gradient descent method with constant step length to solve the dual formulation of the ROF image restoration model. The ROF model is a technique for removing noise from images by minimizing the total variation of the image subject to a fidelity constrain.

The function takes as input variables:

- w1, w2: Dual variables, initial guess.
- f: Noisy image
- lbd: Constant fidelity parameter
- alpha: Fixed step length
- NIT: Maximum number of iterations
- GapTol: Convergence tolerance (stop criterion) for relative duality gap
- verbose: A flag to print intermediate results

The function returns as output variables:

- u: Primal variable, numerical solution - restored image
- w = (w1,w2): Dual variable, numerical solution
- Energy: The value of the objective function
- DGap: Duality Gap
- TimeCost: CPU time cost, iteration by iteration
- itr: Number of iterations

The function works as follows:

- It initializes some variables such as g, gx, gy, sf, which are related to the noisy image f and the fidelity parameter lbd.
- It computes the initial energy, which is the objective function value of the dual problem. The objective function is the squared norm of the divergence of w minus lambda times f.
- It computes the initial primal variable u and the initial duality gap. The primal variable is obtained by subtracting f from the divergence of w divided by lambda. The duality gap is obtained by summing the norm of the gradient of u and the inner product of u and w.

It starts a loop for a maximum of NIT iterations or until the relative duality gap is less than GapTol.

- It computes the gradient of the objective function with respect to w, which is given by dFx and dFy.
- It performs a semi-implicit gradient descent step with constant step length alpha. This means that it updates w by subtracting alpha times the gradient

and then dividing it by one plus alpha times the norm of the gradient. This step ensures that w stays in the feasible set, which is the unit ball.

- It computes the new energy value using the updated w.
- It computes the new primal variable u and the new duality gap using the updated w.
- It tests for convergence by checking if the relative duality gap is less than GapTol. If so, it exits the loop. Otherwise, it continues to the next iteration.

Chambolle's method is a variant of the gradient projection method that avoids explicit projections onto the feasible set by using a semi-implicit update rule. It is an efficient and simple method that converges linearly to a minimizer of the ROF model.

Based on the paper and the search results, I can say that the main difference between the results and performance of denoising of Chambolle and GPCL is that GPCL is faster and more accurate than Chambolle. The paper shows that GPCL converges in fewer iterations and has smaller errors than Chambolle for various test images. The search results also confirm that Chambolle's algorithm may converge slowly.

If we add Gaussian noise (mean=0 , std=0.1) to original image and choose parameter like this:

```
lbd = 16; % fidelity parameter
alpha = 0.1; % fixed step length
NIT = 100; % maximum number of iterations
GapTol = 1e-4; % convergence tolerance for relative duality
gap
verbose = true; % display iteration information or not
```

The result will be like this:

It seems that Chambolle's performance is better than GPCL due to PSNR metric.

Question 3:

A)

This function is called ***isodiff*** and it performs isotropic diffusion on a 2D grey-scale image. Isotropic diffusion is a process of smoothing an image by averaging the pixel values with their neighbors. The function takes three parameters: im, lambda and constant.

- im is the input image that must be 2D and grey-scale. The function will throw an error if the image has more than two dimensions.

- lambda is a scalar value that controls the diffusion rate. It must be positive and small enough to avoid numerical instability. A typical value is 0.25.

- constant is a scalar value that controls the conduction coefficient. It must be positive and determines how much diffusion occurs across edges. A larger value means more smoothing and less edge preservation.

The function returns a matrix diff that has the same size as the input image im and contains the diffused image. The function uses a finite difference scheme to approximate the partial differential equation of isotropic diffusion. It computes the differences between the pixel values and their four neighbors (north, south, east and west) and multiplies them by the conduction coefficient. Then it adds these differences to the original pixel values, scaled by the diffusion rate lambda. The function pads the input image with zeros to avoid boundary issues.

The function can be called in a loop to perform multiple iterations of diffusion until a desired level of smoothing is achieved.

This function is called ***anisodiff*** and it performs anisotropic diffusion on a 2D grey-scale image. Anisotropic diffusion is a process of smoothing an image by averaging the pixel values with their neighbors, but only in the direction of low gradients. This means that the diffusion is more selective and preserves edges better than isotropic diffusion. The function takes five parameters: im, niter, kappa, lambda and option.

- im is the input image that must be 2D and grey-scale. The function will throw an error if the image has more than two dimensions.

- niter is the number of iterations of diffusion to perform. It must be a positive integer and determines how much smoothing is applied.

- kappa is a scalar value that controls the sensitivity of the diffusion. It must be positive and determines the threshold for edge detection. A larger value means less smoothing and more edge preservation.

- lambda is a scalar value that controls the diffusion rate. It must be positive and small enough to avoid numerical instability. A typical value is 0.25.

- option is a scalar value that controls the conduction function. It can be either 1 or 2 and determines how the conduction coefficient varies with the gradient. Option 1 uses an exponential function and option 2 uses an inverse quadratic function.

The function returns a matrix diff that has the same size as the input image im and contains the diffused image. The function uses a finite difference scheme to approximate the partial differential equation of anisotropic diffusion. It computes the differences between the pixel values and their four neighbors (north, south, east and west) and multiplies them by the conduction coefficient, which depends on the option and kappa parameters. Then it adds these differences to the original pixel values, scaled by the diffusion rate lambda. The function pads the input image with zeros to avoid boundary issues.

B)

SSIM:

It is a method for predicting the perceived quality of digital images and videos by measuring the similarity between two images. It considers image degradation as perceived change in structural information, while also incorporating important perceptual phenomena, such as luminance masking and contrast masking. It is a full reference metric, which means it needs an original image as a reference to compare with a distorted image. It is based on four components: mean, variance, covariance and luminance. The formula for SSIM is:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

where x and y are two images, μx and μy are their means, σx and σy are their standard deviations, σxy is their covariance, and c1 and c2 are two constants to avoid instability when the denominator is close to zero. A higher SSIM index indicates better perceptual quality.

NIQE:

Naturalness image quality evaluator (NIQE) is a no-reference image quality metric that assesses the visual impact of natural scene statistics (NSS) based image features. It does not require any training on human-rated distorted images, and it can measure the perceptual quality of any image without knowing the distortion type. NIQE compares the image features to a default model computed from images of natural scenes, or a custom model trained on a specific image dataset. A smaller score indicates better perceptual quality. NIQE is based on the assumption that natural images have some statistical regularities that are consistent across different scenes.

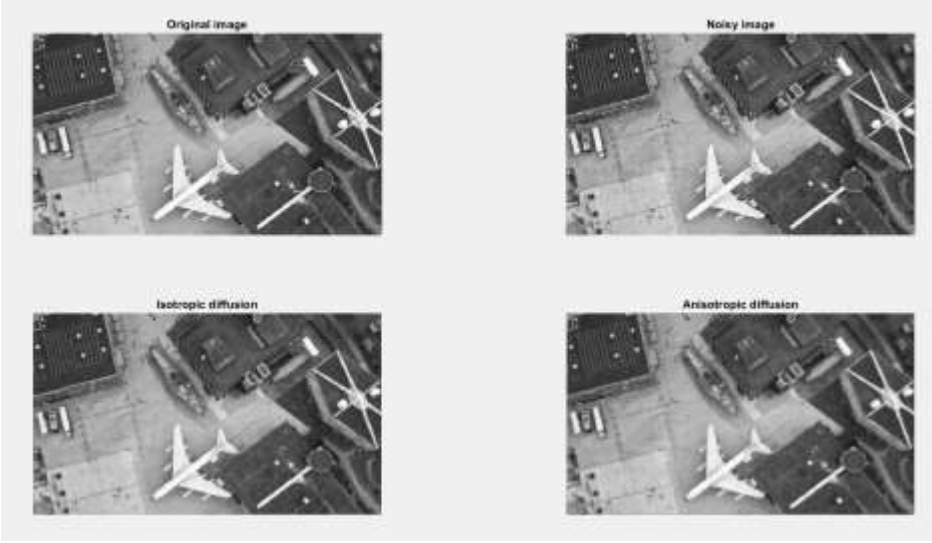
If we add Gaussian noise (mean=0 , std=0.05) to original image and choose parameter like this:

```
% Denoise using isotropic diffusion

lambda = 0.1; % adjust the diffusion speed here
constant = 1.8; % adjust the conduction coefficient here

% Denoise using anisotropic diffusion
niter = 2; % adjust the number of iterations here
kappa = 25; % adjust the conduction coefficient here
lambda = 0.18; % adjust the diffusion speed here
option = 2; % choose between diffusion equation 1 or 2
```

The results of code are in the below:



	NIQE	SSIM
Original	2.8057	NaN
Noisy	10.524	0.61949
Isotropic	4.3753	0.79513
Anisotropic	5.1498	0.80686

As you can see, the image is more blurred with the Anisotropic filter and it is more natural with the Isotropic filter, so the NIQE is lower for the Isotropic filter, but the Anisotropic removes the image noise better and the SSIM is higher for it.

C)

Anisotropic and isotropic filters are different from Gaussian filters in the following ways:

- Anisotropic filters use a space-variant filter that depends on the local content of the image, while isotropic filters use a space-invariant filter that is the same for all pixels. Gaussian filters are a type of isotropic filters that use a Gaussian function as the filter.
- Anisotropic filters can achieve maximum diffusion while preserving strong edges in the image by using weighted averaging and optimizing the weights based on the local neighborhood variances. Isotropic filters, including Gaussian filters, use a uniform averaging that blurs the image uniformly and may lose some details.
- Anisotropic filters are non-linear and space-variant transformations of the original image, while isotropic filters, including Gaussian filters, are linear and space-invariant transformations of the original image.
- Anisotropic filters can be implemented by using different conduction methods, such as exponential or quadratic, that control how the diffusion process behaves near edges or regions of high contrast. Isotropic filters, including Gaussian filters, can be implemented by convolution with a filter function or by using a discrete kernel.