Project 2 - PnP with CNN denoisers, DIP, and SURE

Yoav Ellinson 206036949
Ofir Miran 206564189

Spring 2025

All the code for this project can be found here:
`https://github.com/yoavellinson/signals_and_images_course/tree/main/project_2`

1 Project questions

Q1. In this section we were asked to modify our previous code for the PNP-ADMM deblurring problem, where the modification is to use the IRCNN model as the denoiser. Besides this change all remains the same. We used the best hyper parameters for the BM3D denoiser as a baseline and tried to improve the results using the new DNN as the denoiser. Recall the hyper-parameters we use:

- σ - the noise level for the denoiser.
- ρ - the penalty parameter.
- η - the linear reduction parameter for σ e.g $\sigma^{(k)} = \eta \cdot \sigma^{(k-1)}$: where k is the iteration index
- γ - the linear increasing parameter for ρ e.g $\rho^{(k)} = \gamma \cdot \rho^{(k-1)}$

We started with the baseline hyper-parameters, chose the 3 examples with the lowest output PSNR and started tuning the hyper-parameters by grid searching for the best combination. The results are: $\sigma = 0.11$,ρ =0.008,η =0.99,$\gamma = 1.01$. Table 1 holds the results for all the test-set images including the avarage PSNR at the output.

Picture	Input PSNR	BM3D Output PSNR	IRCNN Output PSNR
1 Cameraman256	21.0	28.92	29.13
2 house	24.23	34.13	34.03
3 peppers256	21.26	31.16	31.35
4 Lena512	25.5	33.83	33.9
5 barbara	22.19	27.44	27.55
6 boat	23.68	31.03	31.04
7 hill	25.14	31.11	31.12
8 couple	23.64	30.77	30.81
Average	**23.33**	**31.051**	**31.116**

Table 1: A summary of PSNR [dB] results for deblurring using BM3D and IRCNN as denoisers

We can see some improvement in PSNR when using the IRCNN DNN as the denoiser, in our opinion the major improvement is in run time,where the each iteration took much less time using the same processor (using a GPU made it even faster). Visual results can be seen at 1

Q2. In order to run the code for section 2 it is required to place the code inside the folder of: `https://github.com/DmitryUlyanov/deep-image-prior`

a) In this section we were asked to adjust a code of Deep Image Prior (DIP) algorithm from `https://github.com/DmitryUlyanov/deep-image-prior`. The code should run over the 8 test images of project 1 and increase the averaged PSNR. We were ask to demonstrate the phenomenon of the DIP algorithm- that running the optimizer too many iterations will make it overfit the noise, and will decrease the PSNR.

The hyper parameter we used are:

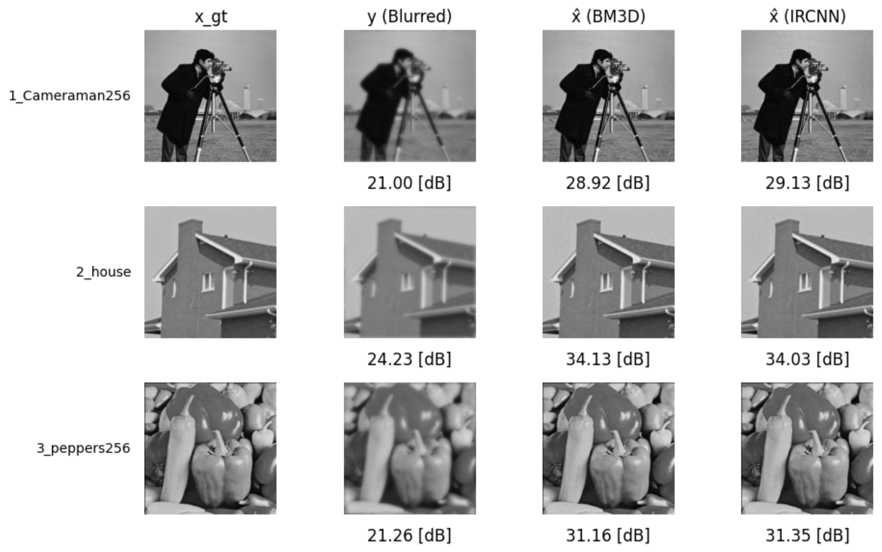

Figure 1: Visual results of the deblurring,output PSNR below each image

- σ - the noise level for the denoiser = 0.1
- number of iteration = 4000
- μ - Learning rate = 0.01
- Optimizer = ADAM
- Regularization noise standard deviation = 1/30

The training graph of avarage PSNR with MSE loss as a function of the number of iterations is in Fig 2

Maximum average PSNR: 27.14 dB (achieved at iteration 2288), results in Table 2, visual examples in Fig 3.

Picture	**Output PSNR**
1 Cameraman256	26.90
2 house	29.17
3 peppers256	26.84
4 Lena512	29.40
5 barbara	24.10
6 boat	27.09
7 hill	27.07
8 couple	26.55
Average	**27.14**

Table 2: PSNR [dB] for each image at this peak iteration (2288)

b) In this section we were asked to use the SURE loss in the DIP algorithem.

The hyper parameter we used are:

- σ - the noise level for the denoiser = 0.1
- number of iteration = 4000
- μ - Learning rate = 0.001
- Optimizer = ADAM
- Regularization noise standard deviation = 1

The training graph of avarage PSNR with SURE loss as a function of the number of iterations is in Fig 4

Maximum average PSNR: 27.64 dB (achieved at iteration 2585). Results in Table 3 and visual results in Fig 5

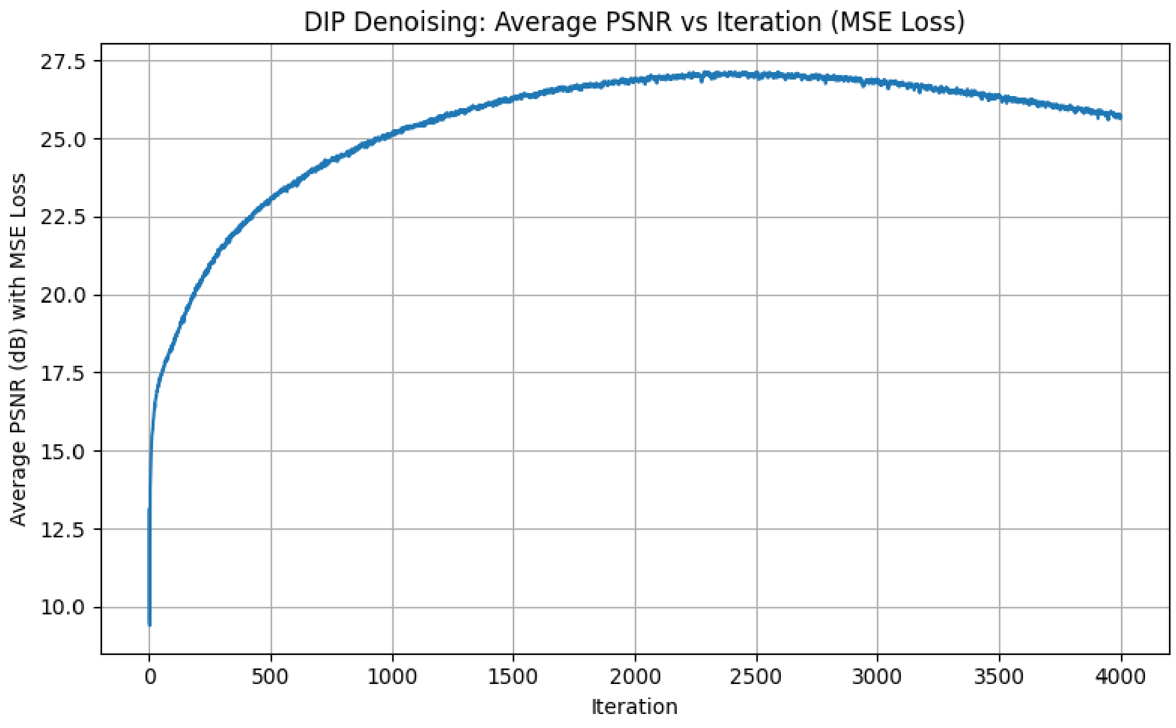

Figure 2: DIP Denoising - Avarage PSNR VS Iteration with MSE loss

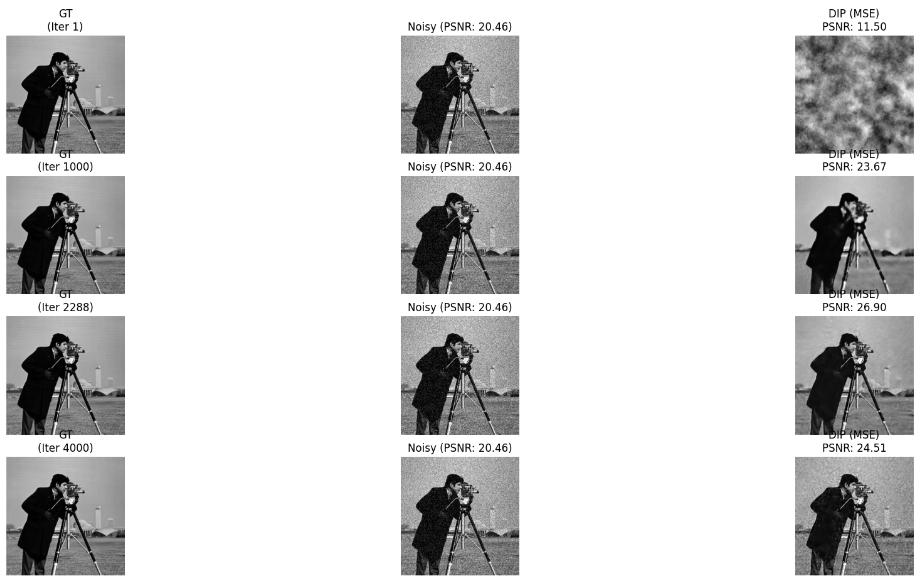

Figure 3: Visual examples of the denoising in different iterations

In conclusion, we got better average PSNR using the SURE loss (27.64 dB), compared to the MSE loss (27.26 dB). It is support the theory, because for Gaussian denoising SURE is generally expected to yield better results than standard MSE loss. This is because SURE provides an unbiased estimate of the true Mean Squared Error to the unknown clean image, allowing the network to directly minimize the actual reconstruction error without needing the Ground Truth itself. Standard MSE, in contrast, requires explicit access to the Ground Truth during training.

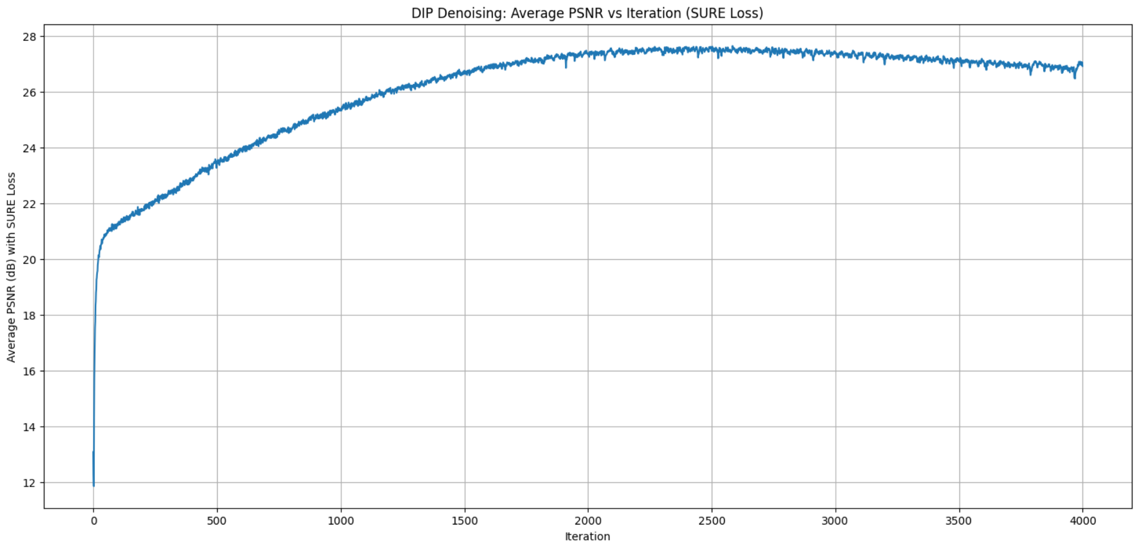

Figure 4: DIP Denoising - Avarage PSNR VS Iteration with SURE loss

Picture	**Output PSNR**
1 Cameraman256	27.41
2 house	29.75
3 peppers256	26.93
4 Lena512	29.60
5 barbara	24.97
6 boat	27.71
7 hill	27.52
8 couple	27.24
Average	**27.64**

Table 3: PSNR [dB] for each image at this peak iteration (2585)

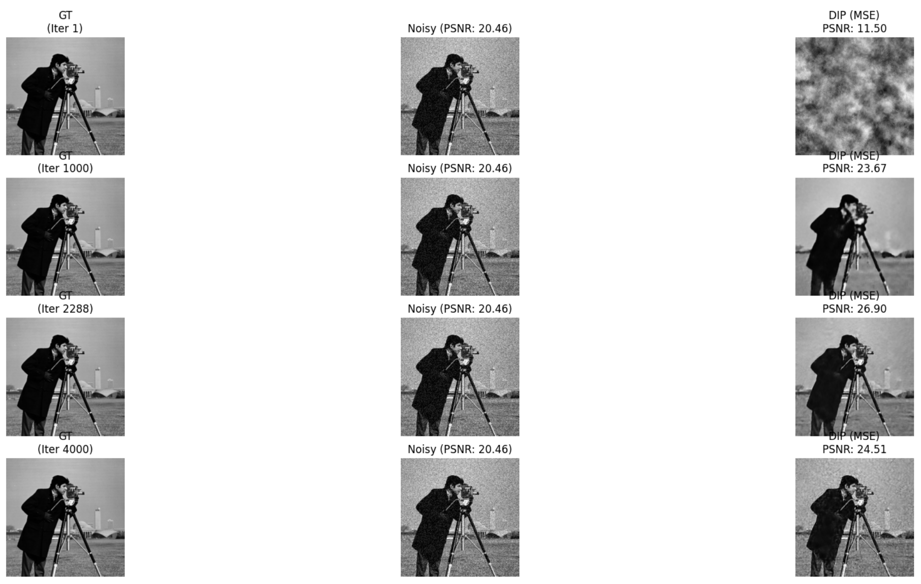

Figure 5: Visual examples of the denoising in different iterations

Appendix 1: Code for image deblurring

```python
import numpy as np
from bm3d import bm3d
import matplotlib.pyplot as plt
from tqdm import tqdm
from ircnn import IRCNN
import torch

np.random.seed(0)

def psnr(x, x_gt):
    """
    Compute the Peak Signal-to-Noise Ratio (PSNR) between two images.

    Inputs:
        x      : Estimated image (numpy array, float32, values in [0, 1])
        x_gt   : Ground truth image (same size and type as x)

    Output:
        PSNR value in decibels (float)
    """
    mse = np.mean((x - x_gt) ** 2)
    return 10 * np.log10(1 / mse)

def add_noise(img, sigma_e):
    """
    Add Gaussian noise to an image with fixed random seed for reproducibility.

    Inputs:
        img      : Clean image (numpy array, float32, values in [0, 1])
        sigma_e : Standard deviation of Gaussian noise

    Output:
        Noisy image y = x_gt + e (numpy array, float32)
    """
    noise = np.random.normal(0, sigma_e, img.shape)
    return img + noise
def cconv2_invAAt_by_fft2_numpy(A,B,eta=0.01):
    # assumes that A (2D image) is bigger than B (2D kernel)

    m, n = A.shape
    mb, nb = B.shape
    # Pad kernel to image size
    bigB = np.zeros_like(A)
    bigB[:mb, :nb] = B

    # Roll to center the PSF
    bigB = np.roll(bigB, shift=(-mb // 2, -nb // 2), axis=(0, 1))

    # FFT of kernel and input
    fft2B = np.fft.fft2(bigB)

    fft2B_norm2 = np.abs(fft2B)**2
    inv_fft2B_norm = 1 / (fft2B_norm2 + eta)

    result = np.real(np.fft.ifft2(np.fft.fft2(A) * inv_fft2B_norm))

    return result



# y_k = AtA_add_eta_inv(At(y) + rho(x -u))

def cconv2_by_fft2_numpy(A, B,flag_conjB=False, eta=1e-2):
    """
    Circular 2D convolution or deconvolution using FFT (NumPy version).

    Args:
        A (np.ndarray): 2D input image (H x W)
        B (np.ndarray): 2D kernel (h x w)
        flag_invertB (bool): If True, performs deconvolution
        eta (float): Regularization parameter for deconvolution

    Returns:
        np.ndarray: Output after convolution or deconvolution
    """
    m, n = A.shape
    mb, nb = B.shape

    # Pad kernel to image size
    bigB = np.zeros_like(A)
    bigB[:mb, :nb] = B

    # Roll to center the PSF
    bigB = np.roll(bigB, shift=(-mb // 2, -nb // 2), axis=(0, 1))

    # FFT of kernel and input
    fft2B = np.fft.fft2(bigB)
    fft2A = np.fft.fft2(A)

    if flag_conjB:
        # Tikhonov regularization for inverse filtering
        fft2B = np.conj(fft2B)# / (np.abs(fft2B)**2 + eta)

    result = np.real(np.fft.ifft2(fft2A * fft2B))

    return result

class PnPADMMDeBlurr:
    def __init__(self,denoiser,max_iter,rho,sigmas,kernel,gamma=1,eta=1,tol=1e-6):
        '''
        denosier (string): denosing function
        max_iter (int): maximum ADMM itterations
        rho(float): ADMM penalty parameter
        sigmas (list): [sigma_psd] for bm3d denoiser and [sigma_r,sigma_s] for BF
                                                    denosier
        '''
        self.kernel = kernel
        self.denoiser = denoiser
        self.max_iter = max_iter
        self.rho = rho
        self.sigmas = sigmas
        self.gamma = gamma
        self.eta=eta
        self.tol = tol
        if denoiser == 'ircnn':
            # self.model = IRCNN(in_nc=1,out_nc=1,nc=64)
            self.model25 = torch.load('/home/workspace/yoavellinson/
                                                      signals_and_images_course/
                                                      project_2/ircnn_gray.pth')
            # current_idx = min(int(np.ceil(sigmas[0] * 255. / 2.) - 1),24)
            # former_idx = 0
            # if current_idx != former_idx:
            #     self.model.load_state_dict(model25[str(current_idx)], strict=True)
            #     self.model.eval()
            #     for _, v in self.model.named_parameters():
            #         v.requires_grad = False
            # #     model = model.to(device)
            # # self.model.load_state_dict(model25,strict=True)
    def get_txt(self):
        return f'rho_{self.rho}_sigma_{self.sigmas},eta_{self.eta}_gamma_{self.gamma}'

    def denoise_sample(self,y):
        if self.denoiser =='bm3d':
            return bm3d(y,sigma_psd=self.sigmas[0])
        elif self.denoiser =='ircnn':
            model = IRCNN(in_nc=1,out_nc=1,nc=64)
            current_idx = min(int(np.ceil(self.sigmas[0] * 255. / 2.) - 1),24)
            former_idx = 0
            if current_idx != former_idx:
                model.load_state_dict(self.model25[str(current_idx)], strict=True)
                model.eval()
                for _, v in model.named_parameters():
                    v.requires_grad = False
            y = torch.tensor(y,dtype=torch.float).unsqueeze(0).unsqueeze(0)
            return model(y).detach().cpu().squeeze().numpy()

    def AtA_add_eta_inv_numpy(self, vec,):
        I = vec.reshape(vec.shape[0], vec.shape[1])

        out = cconv2_invAAt_by_fft2_numpy(I, self.kernel, eta=self.rho)

        return out.reshape(vec.shape[0], -1)

    def At_numpy(self,vec):
        I = vec.reshape(vec.shape[0], vec.shape[1])
        out = cconv2_by_fft2_numpy(I,self.kernel, flag_conjB=True)
        return out.reshape(vec.shape[0], -1)

    def __call__(self, y,img):
        '''
        y: blurred image (2D numpy array)
        Returns: Deblurred image
        '''
        # Initialization
        N = y.shape[0]*y.shape[1]
        #as in the paper

        x_k = y.copy()
        v_k = y.copy()
        u_k = np.zeros_like(y)

        rho_tmp = self.rho
        sigma_tmp = self.sigmas[0]

        i = 0
        res = 10
        pbar = tqdm(total=self.max_iter,desc='Residuals',leave=False)
        psnr_old = 0
        while (res > self.tol) and i < self.max_iter:
            x_k_1,v_k_1,u_k_1 = self.pnp_admm_step(y,x_k,v_k,u_k)
            psnr_mid = psnr(x_k_1,img)
            # print(f'PSNR:{psnr_mid}')
            delta_psnr = psnr_mid - psnr_old
            res_x = (1/np.sqrt(N)) * np.sqrt(np.sum((x_k_1-x_k)**2,axis=(0,1)))
            res_z = (1/np.sqrt(N)) * np.sqrt(np.sum((v_k_1-v_k)**2,axis=(0,1)))
            res_u = (1/np.sqrt(N)) * np.sqrt(np.sum((u_k_1-u_k)**2,axis=(0,1)))
            res = res_u+res_x+res_z
            # if res < 0 and i>10:
            #      break
            if delta_psnr >0:
                self.rho *= self.gamma
            elif delta_psnr<0 and i > 5:
                break
            v_k=v_k_1
            x_k=x_k_1
            u_k=u_k_1
            psnr_old = psnr_mid
            # self.rho*=1.5
            self.sigmas[0] *=self.eta
            i+=1
            pbar.update(1)
            pbar.set_description(f'PSNR={psnr_mid:.8f}')

        self.rho = rho_tmp
        self.sigmas[0] = sigma_tmp
        return x_k

    def prox(self,y,x_tilde):
        a = self.At_numpy(y) + self.rho*(x_tilde)
        return self.AtA_add_eta_inv_numpy(a)

    def pnp_admm_step(self,y,x,v,u):
        x_tilde = v-u

        x = self.prox(y,x_tilde)

        v_tilde = x+u

        v= self.denoise_sample(v_tilde)

        u += x-v

        return x,v,u

def main(denoiser='bm3d'):
    """
    Denoise a set of grayscale images using bilateral filtering,
    compute and print PSNR values, and display visual comparisons.
    """
    image_names = ['1_Cameraman256', '2_house', '3_peppers256', '4_Lena512',
                   '5_barbara', '6_boat', '7_hill', '8_couple']
    #hyper parameters

    max_iter=35
    sigmas = [0.09] if denoiser =='bm3d' else [0.11]
    rho = 0.013 if denoiser =='bm3d' else 0.008

    eta = 0.99 if denoiser =='bm3d' else 0.99
    gamma=1.01 if denoiser =='bm3d' else 1.01

    #blurring kernel

    i = np.arange(-7, 8)
    j = np.arange(-7, 8)
    kernel = np.zeros((len(i),len(j)))
    for ii in range(len(i)):
        for jj in range(len(j)):
            kernel[ii,jj] = 1/(1+i[ii]**2+j[jj]**2)
    kernel /= np.sum(kernel)

    input_psnrs = []
    denoised_psnrs = []
    images_gt = []
    images_noisy = []
    images_denoised = []

    dir_path = './test_set'
    deblurrer = PnPADMMDeBlurr(denoiser=denoiser,max_iter=max_iter,rho=rho,sigmas=sigmas
                                                ,kernel=kernel,eta=eta,gamma=gamma,tol=
                                                      1e-5)
    for name in image_names:
        try:
            img = plt.imread(f'{dir_path}/{name}.png')
        except FileNotFoundError:
            print(f"Error: File {dir_path}+/{name}.png not found.")
            return -1
        except Exception as e:
            print(f"Could not load {name}.png due to error: {e}")
            return -1

        # Convert to grayscale and normalize
        if img.ndim == 3:
            img = np.mean(img, axis=2)  # convert RGB to grayscale
        if img.dtype != np.float32 and img.max() > 1.0:
            img = img.astype(np.float32) / 255.0  # normalize to [0, 1]

        y = cconv2_by_fft2_numpy(img, kernel)
        y = add_noise(y,sigma_e=0.01)
        x_hat = deblurrer(y,img)

        psnr_input = psnr(y, img)
        psnr_output = psnr(x_hat,img)

        input_psnrs.append(psnr_input)
        denoised_psnrs.append(psnr_output)

        print(f"{name}: Input PSNR = {psnr_input:.2f}, Output PSNR = {psnr_output:.2f}")

        # Save for visualization
        images_gt.append(img)
        images_noisy.append(y)
        images_denoised.append(x_hat)

    input_psnr_mean =np.mean(input_psnrs)
    output_psnr_mean =np.mean(denoised_psnrs)

    print("\nAverage Input PSNR: {:.2f}".format(input_psnr_mean))
    print("Average Deblurred PSNR: {:.2f}".format(output_psnr_mean))

    if denoiser =='bm3d' or denoiser=='ircnn':
        sigma_txt = f'sigma={deblurrer.sigmas[0]:.4f}'
    else:
        sigma_txt = f'sigma_s={deblurrer.sigmas[0]:.4f},sigma_r={deblurrer.sigmas[-1]:.
                                                            4f}'



    hyperparams = f'{sigma_txt},rho={rho},eta={eta},gamma={gamma}'
    # ===============================
    #  Plot: x_gt, y (noisy), x   (denoised)
    # ===============================
    fig, axs = plt.subplots(len(image_names), 3, figsize=(10, 2 * len(image_names)))
    fig.suptitle(f'Deblurring Results:Denoiser:{denoiser}\nPSNR-Input={input_psnr_mean:.
                                                  2f},PSNR-Output={output_psnr_mean:.2f}\
                                                  nHyperparams:{hyperparams}', fontsize=16
                                                  )

    for row_idx, name in enumerate(image_names):
        img = images_gt[row_idx]  # Ground truth
        y = images_noisy[row_idx]  # Noisy image
        x_hat = images_denoised[row_idx]  # Denoised output
        for col_idx, image in enumerate([img, y, x_hat]):
            axs[row_idx, col_idx].imshow(image, cmap='gray', vmin=0, vmax=1)
            axs[row_idx, col_idx].axis('off')

            # Set column titles only on first row
            if row_idx == 0:
                if col_idx == 0:
                    axs[row_idx, col_idx].set_title("x_gt")
                elif col_idx == 1:
                    axs[row_idx, col_idx].set_title("y (Blurred)")
                elif col_idx ==2 :
                    axs[row_idx, col_idx].set_title(" x   (Deblurred)")

        # Add image name label on the left of each row
        axs[row_idx, 0].text(-0.1, 0.5, name, fontsize=10, va='center', ha='right',
                             transform=axs[row_idx, 0].transAxes, rotation=0)

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    filename = f'./plots/pnp_admm_results_max_{deblurrer.get_txt()}.png'
    plt.savefig(filename)
    print(f'Saved: {filename}')
    plt.show()

from itertools import product

def run_grid_search():
    sigmas_list_s = [0.1,0.08,0.06]
    # sigmas_list_r = [0.01,0.04,0.08, 0.1,0.5]

    rhos_list = [0.009,0.01,0.1]

    eta_list = [1,0.99,0.999,0.95,0.9]

    gamma_list = [1]

    best_psnr = -np.inf
    best_config = None

    for sigma_s, rho, eta, gamma in tqdm(product(sigmas_list_s, rhos_list, eta_list,
                                                  gamma_list),total=len(sigmas_list_s)*len
                                                  (rhos_list)*len(eta_list)*len(gamma_list
                                                  )):
        print(f"\nRunning: sigma={sigma_s}, rho={rho}, reduce_sigma={eta}, increase_rho=
                                                      {gamma}")
        denoiser = 'ircnn'
        max_iter = 25
        kernel = make_kernel()

        deblurrer = PnPADMMDeBlurr(
            denoiser=denoiser,
            max_iter=max_iter,
            rho=rho,
            sigmas=[sigma_s],
            kernel=kernel,
            gamma=gamma,
            eta=eta
        )

        avg_psnr = evaluate_deblurrer(deblurrer)
        print(f'PSNR:{avg_psnr}')
        if avg_psnr > best_psnr:
            best_psnr = avg_psnr
            best_config = (sigma_s, rho, eta, gamma)

    print("\n==== Best Configuration ====")
    print(f"Sigma: {best_config[0]}, Rho: {best_config[1]}, Reduce Sigma: {best_config[2
                                                  ]}, Increase Rho: {best_config[3]}")
    print(f"Average PSNR: {best_psnr:.2f}")

def main_both():
    """
    Denoise a set of grayscale images using bilateral filtering,
    compute and print PSNR values, and display visual comparisons.
    """
    # image_names = ['1_Cameraman256', '2_house']
    image_names = ['1_Cameraman256', '2_house', '3_peppers256', '4_Lena512',
                   '5_barbara', '6_boat', '7_hill', '8_couple']


    #blurring kernel
    i = np.arange(-7, 8)
    j = np.arange(-7, 8)
    kernel = np.zeros((len(i),len(j)))
    for ii in range(len(i)):
        for jj in range(len(j)):
            kernel[ii,jj] = 1/(1+i[ii]**2+j[jj]**2)
    kernel /= np.sum(kernel)

    #hyper parameters
    max_iter=25
    eta = 0.99
    gamma=1.01

    deblurrer_bm3d = PnPADMMDeBlurr(denoiser='bm3d',max_iter=max_iter,rho=0.013,sigmas=[
                                            0.09],kernel=kernel,eta=eta,gamma=gamma,
                                            tol=1e-5)
    deblurrer_ircnn = PnPADMMDeBlurr(denoiser='ircnn',max_iter=max_iter,rho=0.008,sigmas
                                            =[0.11],kernel=kernel,eta=eta,gamma=
                                            gamma,tol=1e-5)



    input_psnrs = []
    denoised_psnrs_bm3d = []
    denoised_psnrs_ircnn = []

    images_gt = []
    images_noisy = []
    images_denoised_bm3d = []
    images_denoised_ircnn = []


    dir_path = './test_set'
    for name in image_names:
        try:
            img = plt.imread(f'{dir_path}/{name}.png')
        except FileNotFoundError:
            print(f"Error: File {dir_path}+/{name}.png not found.")
            return -1
        except Exception as e:
            print(f"Could not load {name}.png due to error: {e}")
            return -1

        # Convert to grayscale and normalize
        if img.ndim == 3:
            img = np.mean(img, axis=2)  # convert RGB to grayscale
        if img.dtype != np.float32 and img.max() > 1.0:
            img = img.astype(np.float32) / 255.0  # normalize to [0, 1]

        y = cconv2_by_fft2_numpy(img, kernel)
        y = add_noise(y,sigma_e=0.01)
        x_hat_bm3d = deblurrer_bm3d(y,img)
        x_hat_ircnn = deblurrer_ircnn(y,img)

        psnr_input = psnr(y, img)
        psnr_output_bm3d = psnr(x_hat_bm3d,img)
        psnr_output_ircnn = psnr(x_hat_ircnn,img)

        input_psnrs.append(psnr_input)
        denoised_psnrs_bm3d.append(psnr_output_bm3d)
        denoised_psnrs_ircnn.append(psnr_output_ircnn)

        print(f"{name}: Input PSNR = {psnr_input:.2f}, Output PSNR BM3D = {
                                                psnr_output_bm3d:.2f} Output PSNR
                                                IRCNN = {psnr_output_ircnn:.2f}")

        # Save for visualization
        images_gt.append(img)
        images_noisy.append(y)
        images_denoised_bm3d.append(x_hat_bm3d)
        images_denoised_ircnn.append(x_hat_ircnn)

    input_psnr_mean =np.mean(input_psnrs)
    output_psnr_mean_bm3d =np.mean(denoised_psnrs_bm3d)
    output_psnr_mean_ircnn =np.mean(denoised_psnrs_ircnn)

    print("\nAverage Input PSNR: {:.2f}".format(input_psnr_mean))
    print(f"Average Deblurred PSNR BM3D/IRCNN: {output_psnr_mean_bm3d}/{
                                                output_psnr_mean_ircnn}")



    # ==============================
    #  Plot: x_gt, y (noisy), x  (denoised)
    # ==============================
    fig, axs = plt.subplots(len(image_names), 4, figsize=(10, 2 * len(image_names)))
    fig.suptitle(f'Deblurring Results PSNR BM3D/IRCNN: {output_psnr_mean_bm3d:.2f}/{
                                                output_psnr_mean_ircnn:.2f}[dB]',
                                                fontsize=16)

    for row_idx, name in enumerate(image_names):
        img = images_gt[row_idx]  # Ground truth
        y = images_noisy[row_idx]  # Noisy image
        x_hat_bm3d = images_denoised_bm3d[row_idx]  # Denoised output
        x_hat_ircnn = images_denoised_ircnn[row_idx]  # Denoised output

        for col_idx, image in enumerate([img, y, x_hat_bm3d,x_hat_bm3d]):
            axs[row_idx, col_idx].imshow(image, cmap='gray', vmin=0, vmax=1)
            axs[row_idx, col_idx].axis('off')
            # Set column titles only on first row
            if row_idx == 0:
                if col_idx == 0:
                    axs[row_idx, col_idx].set_title("x_gt")
                elif col_idx == 1:
                    axs[row_idx, col_idx].set_title("y (Blurred)")
                    axs[row_idx, col_idx].text(0.5, -0.1,
                                           f'{input_psnrs[row_idx]:.2f} [dB]',
                                           transform=axs[row_idx, col_idx].transAxes,
                                           ha='center', va='top', fontsize=12)
                elif col_idx ==2 :
                    axs[row_idx, col_idx].set_title(" x  (BM3D)")
                    axs[row_idx, col_idx].text(0.5, -0.1,
                                           f'{denoised_psnrs_bm3d[row_idx]:.2f} [dB]',
                                           transform=axs[row_idx, col_idx].transAxes,
                                           ha='center', va='top', fontsize=12)
                elif col_idx ==3 :
                    axs[row_idx, col_idx].set_title(" x  (IRCNN)")
                    axs[row_idx, col_idx].text(0.5, -0.1,
                                           f'{denoised_psnrs_ircnn[row_idx]:.2f} [dB]',
                                           transform=axs[row_idx, col_idx].transAxes,
                                           ha='center', va='top', fontsize=12)
            else:
                if col_idx == 1:
                    axs[row_idx, col_idx].text(0.5, -0.1,
                                           f'{input_psnrs[row_idx]:.2f} [dB]',
                                           transform=axs[row_idx, col_idx].transAxes,
                                           ha='center', va='top', fontsize=12)
                elif col_idx ==2 :
                    axs[row_idx, col_idx].text(0.5, -0.1,
                                           f'{denoised_psnrs_bm3d[row_idx]:.2f} [dB]',
                                           transform=axs[row_idx, col_idx].transAxes,
                                           ha='center', va='top', fontsize=12)
                elif col_idx ==3 :
                    axs[row_idx, col_idx].text(0.5, -0.1,
                                           f'{denoised_psnrs_ircnn[row_idx]:.2f} [dB]',
                                           transform=axs[row_idx, col_idx].transAxes,
                                           ha='center', va='top', fontsize=12)
        # Add image name label on the left of each row
        axs[row_idx, 0].text(-0.1, 0.5, name, fontsize=10, va='center', ha='right',
                             transform=axs[row_idx, 0].transAxes, rotation=0)

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    filename = f'./plots/pnp_admm_results_both.png'
    plt.savefig(filename)
    print(f'Saved: {filename}')
    plt.show()

from itertools import product

def run_grid_search():
    sigmas_list_s = [0.1,0.08,0.06]
    # sigmas_list_r = [0.01,0.04,0.08, 0.1,0.5]

    rhos_list = [0.009,0.01,0.1]

    eta_list = [1,0.99,0.999,0.95,0.9]

    gamma_list = [1]

    best_psnr = -np.inf
    best_config = None

    for sigma_s, rho, eta, gamma in tqdm(product(sigmas_list_s, rhos_list, eta_list,
                                              gamma_list),total=len(sigmas_list_s)*len
                                              (rhos_list)*len(eta_list)*len(gamma_list
                                              )):
        print(f"\nRunning: sigma={sigma_s}, rho={rho}, reduce_sigma={eta}, increase_rho=
                                                  {gamma}")
        denoiser = 'ircnn'
        max_iter = 25
        kernel = make_kernel()

        deblurrer = PnPADMMDeBlurr(
            denoiser=denoiser,
            max_iter=max_iter,
            rho=rho,
            sigmas=[sigma_s],
            kernel=kernel,
            gamma=gamma,
            eta=eta
        )

        avg_psnr = evaluate_deblurrer(deblurrer)
        print(f'PSNR:{avg_psnr}')
        if avg_psnr > best_psnr:
            best_psnr = avg_psnr
            best_config = (sigma_s, rho, eta, gamma)

    print("\n==== Best Configuration ====")
    print(f"Sigma: {best_config[0]}, Rho: {best_config[1]}, Reduce Sigma: {best_config[2
                                              ]}, Increase Rho: {best_config[3]}")
    print(f"Average PSNR: {best_psnr:.2f}")

def make_kernel():
    i = np.arange(-7, 8)
    j = np.arange(-7, 8)
    kernel = np.zeros((len(i), len(j)))
    for ii in range(len(i)):
        for jj in range(len(j)):
            kernel[ii, jj] = 1 / (1 + i[ii] ** 2 + j[jj] ** 2)
    return kernel / np.sum(kernel)

def evaluate_deblurrer(deblurrer):
    image_names = ['1_Cameraman256','5_barbara','3_peppers256']
    dir_path = './test_set'
    input_psnrs = []
    denoised_psnrs = []

    for name in image_names:
        try:
            img = plt.imread(f'{dir_path}/{name}.png')
        except Exception as e:
            print(f"Failed to load {name}: {e}")
            return -1

        if img.ndim == 3:
            img = np.mean(img, axis=2)
        if img.dtype != np.float32 and img.max() > 1.0:
            img = img.astype(np.float32) / 255.0

        y = cconv2_by_fft2_numpy(img, deblurrer.kernel)
        y = add_noise(y, sigma_e=0.01)
        x_hat = deblurrer(y,img)
        psnr_output = psnr(img, x_hat)
        denoised_psnrs.append(psnr_output)

    return np.mean(denoised_psnrs)


if __name__ == "__main__":
    # main(denoiser='BL') #for bilateral filter denoiser
    # main(denoiser='ircnn')
    # run_grid_search()
    main_both()
```

Appendix 2: Code for image denoising

```
    ### Libraries ###
import os
import numpy as np
import matplotlib.pyplot as plt
import torch
from models import *
from utils.denoising_utils import *
from skimage.metrics import peak_signal_noise_ratio as compare_psnr

### Hyper Parameters ###
num_iter = 4000  # Max iterations for training (ensure it's long enough to see
                                             overfitting)
OPTIMIZER = 'adam'

# Loss Function Flag
USE_SURE_LOSS = True  # Set to True for SURE, False for MSE
# --- Adjust LR and reg_noise_std based on current loss type ---
if USE_SURE_LOSS:
    current_LR = 0.001
    current_reg_noise_std = 1.0
    current_loss_type_name = "SURE"
else:
    current_LR = 0.01
    current_reg_noise_std = 1. / 30.
    current_loss_type_name = "MSE"

### Parameters ###
imsize = -1
PLOT = False  # We will handle plotting separately at the end
sigma = 0.1  # standard deviation of Gaussian noise
show_every = 100  # Print progress every X iterations
exp_weight = 0.99  # For averaging output
input_depth = 1
OPT_OVER = 'net'
INPUT = 'noise'
pad = 'reflection'

# List of test images
image_files = ['1_Cameraman256.png', '2_house.png', '3_peppers256.png', '4_Lena512.png',
               '5_barbara.png', '6_boat.png', '7_hill.png', '8_couple.png']
image_files = [os.path.join('test_set/', f) for f in image_files]

### Settings ###
torch.backends.cudnn.enabled = True
torch.backends.cudnn.benchmark = True

if torch.cuda.is_available():
    device = torch.device("cuda")
    dtype = torch.cuda.FloatTensor
    print("Using CUDA GPU")
else:
    device = torch.device("cpu")
    dtype = torch.FloatTensor
    print("Using CPU")
```

```
# Visualization Specific Settings
VISUAL_EXAMPLES_IMAGE_FNAME = '1_Cameraman256.png'  # The image to show detailed visual
                                            examples for

# Dictionary to store *all* iteration results for the selected image
# Format: {iteration: {'out_np': ..., 'psnr': ...}}
full_history_for_selected_image = {}

# This will be populated after training to select specific points
dynamic_visual_iterations_global = []

# Store PSNRs for all images to find average peak
psnrs_per_iter_all_images = []

### Main Code ###
print(f"\n--- Starting Training with {current_loss_type_name} Loss ---")
print(f"Loss: {current_loss_type_name}, LR: {current_LR}, reg_noise_std: {
                                            current_reg_noise_std}")

for fname in image_files:
    print(f"\nProcessing {os.path.basename(fname)} with {current_loss_type_name} Loss")

    # Fix random seed for reproducibility per image
    np.random.seed(0)
    torch.manual_seed(0)

    # Load GT image
    img_pil = crop_image(get_image(fname, imsize)[0].convert('L'), d=32)
    img_np = pil_to_np(img_pil)

    # Add Gaussian noise
    noise = np.random.normal(0, sigma, img_np.shape).astype(np.float32)
    img_noisy_np = img_np + noise
    img_noisy_np = np.clip(img_noisy_np, 0, 1)

    # Convert to torch
    img_noisy_torch = np_to_torch(img_noisy_np).type(dtype)

    # Initialize DIP network
    net = get_net(input_depth, 'skip', pad,
                  skip_n33d=128, skip_n33u=128, skip_n11=4, num_scales=5,
                  upsample_mode='bilinear',
                  n_channels=1).type(dtype)

    net_input_saved = None
    noise_input = None

    if USE_SURE_LOSS:
        net_input_saved = img_noisy_torch.detach().clone()
        noise_input = torch.zeros_like(img_noisy_torch).type(dtype)
    else:
        net_input = get_noise(input_depth, INPUT, (img_pil.size[1], img_pil.size[0])).
                                                type(dtype).detach()
        net_input_saved = net_input.detach().clone()
        noise_input = net_input.detach().clone()

    mse_loss_fn = torch.nn.MSELoss().type(dtype)

    out_avg_wrapper = [None]
    psnrs_wrapper = []


    def closure():
        # Zero gradients for network parameters
        for param in net.parameters():
            if param.grad is not None:
                param.grad.zero_()

        current_net_input = None

        if USE_SURE_LOSS:
            # When using SURE, net_input is based on the noisy image.
            current_net_input = net_input_saved.clone()
            if current_reg_noise_std > 0:
                current_net_input += (torch.randn_like(noise_input) *
                                                            current_reg_noise_std)

            current_net_input.requires_grad_(True)

            if current_net_input.grad is not None:
                current_net_input.grad.zero_()

            out = net(current_net_input)

            # SURE Loss: ||out - noisy_image||^2 + 2 * sigma^2 * div(out) - sigma^2 * N

            # Find the divergence
            divergence_term = torch.autograd.grad(out, current_net_input,
                                                  grad_outputs=torch.ones_like(out),
                                                  retain_graph=True, allow_unused=True)[
                                                                                              0
                                                                                              ]

            divergence_term_sum = divergence_term.sum()

            # SURE Loss
            loss = torch.norm(out - img_noisy_torch) ** 2 + \
                   2 * (sigma ** 2) * divergence_term_sum - \
                   (sigma ** 2) * img_noisy_torch.numel()

        else:  # Use MSE Loss
            if current_reg_noise_std > 0:
                current_net_input = net_input_saved + (noise_input.normal_() *
                                                            current_reg_noise_std)
            else:
                current_net_input = net_input_saved

            out = net(current_net_input)
            loss = mse_loss_fn(out, img_noisy_torch)

        loss.backward()

        if out_avg_wrapper[0] is None:
            out_avg_wrapper[0] = out.detach()
        else:
            out_avg_wrapper[0] = out_avg_wrapper[0] * exp_weight + out.detach() * (1 -
                                                        exp_weight)

        out_np = torch_to_np(out.detach())
        psnr_gt = compare_psnr(img_np, out_np)
        psnrs_wrapper.append(psnr_gt)

        # Store full history for the selected image
        current_iter_num = len(psnrs_wrapper)
        if os.path.basename(fname) == VISUAL_EXAMPLES_IMAGE_FNAME:
            full_history_for_selected_image[current_iter_num] = {
                'out_np': out_np.copy(),
                'psnr': psnr_gt
            }

        # Iterations status
        if len(psnrs_wrapper) % show_every == 0:
            print(f"Iter {len(psnrs_wrapper):04d} | PSNR_GT: {psnr_gt:.2f} | Loss: {loss
                                                        .item():.4f}")

        return loss


    p = get_params(OPT_OVER, net, net_input_saved if not USE_SURE_LOSS else None)
    optimize(OPTIMIZER, p, closure, current_LR, num_iter)
    psnrs_per_iter_all_images.append(psnrs_wrapper)

# Final Analysis
min_len_all_images = min(len(p) for p in psnrs_per_iter_all_images)
psnrs_per_iter_all_images = [p[:min_len_all_images] for p in psnrs_per_iter_all_images]
avg_psnr_across_all_images = np.mean(psnrs_per_iter_all_images, axis=0)
```

```
max_avg_psnr_iter_idx = np.argmax(avg_psnr_across_all_images)
max_avg_psnr_value = avg_psnr_across_all_images[max_avg_psnr_iter_idx]

print(f"\n--- {current_loss_type_name} Loss Summary (All Images) ---")
print(f"Maximum average PSNR: {max_avg_psnr_value:.2f} dB (achieved at iteration {
                                            max_avg_psnr_iter_idx + 1}).")
print("PSNR for each image at this peak iteration:")

# Plotting
for i, fname in enumerate(image_files):
    image_name = os.path.basename(fname)
    psnr_at_max_avg = psnrs_per_iter_all_images[i][max_avg_psnr_iter_idx]
    print(f"- {image_name}: {psnr_at_max_avg:.2f} dB")

plt.figure(figsize=(8, 5))
plt.plot(avg_psnr_across_all_images)
plt.xlabel("Iteration")
plt.ylabel(f"Average PSNR (dB) with {current_loss_type_name} Loss")
plt.title(f"DIP Denoising: Average PSNR vs Iteration ({current_loss_type_name} Loss)")
plt.grid(True)
plt.tight_layout()
plt.savefig(f"dip_avg_psnr_vs_iteration_{current_loss_type_name.lower()}.png")
plt.show()

# Determine specific visualization iterations
peak_iter = max_avg_psnr_iter_idx + 1

dynamic_visual_iterations_global = []

# 1. The first iteration (1-indexed)
if 1 <= num_iter:
    dynamic_visual_iterations_global.append(1)

# 2. Iteration 1000, if available
if 1000 <= num_iter:
    dynamic_visual_iterations_global.append(1000)
else:  # If num_iter is less than 1000 but more than 1, add a middle point as a fallback
    if num_iter > 1 and (num_iter // 2) not in dynamic_visual_iterations_global:
        dynamic_visual_iterations_global.append(num_iter // 2)

# 3. The best iteration (peak_iter)
if peak_iter not in dynamic_visual_iterations_global and peak_iter <= num_iter:
    dynamic_visual_iterations_global.append(peak_iter)

# 4. The last iteration
if num_iter not in dynamic_visual_iterations_global:
    dynamic_visual_iterations_global.append(num_iter)

# Ensure unique and sorted iterations
dynamic_visual_iterations_global = sorted(list(set(dynamic_visual_iterations_global)))

# Filter out iterations for which we do not have data (e.g., if num_iter is very small
                                            or iterations were skipped)
dynamic_visual_iterations_global = sorted([
    iter_val for iter_val in dynamic_visual_iterations_global
    if iter_val in full_history_for_selected_image
])

visual_data_for_selected_image = {
    iter_val: full_history_for_selected_image[iter_val]
    for iter_val in dynamic_visual_iterations_global
}

print(f"\nSelected iterations for visual examples: {dynamic_visual_iterations_global}")


# Plot Visual Examples for the selected image
def plot_visual_examples(gt_img_np, noisy_img_np, visual_data, iterations_to_plot,
                                            image_title, loss_type):
    num_iterations = len(iterations_to_plot)
    if num_iterations == 0:
        print("No iterations to plot for visual examples.")
        return

    fig, axes = plt.subplots(num_iterations, 3, figsize=(12, 4 * num_iterations))

    # Ensure axes is always 2D for consistent indexing
    if num_iterations == 1:
        axes = np.array([axes])

    for i, iteration in enumerate(iterations_to_plot):
        # GT Image - Squeeze the channel dimension (1) to make it (H, W)
        axes[i, 0].imshow(gt_img_np.squeeze(), cmap='gray')
        axes[i, 0].set_title(f"GT\n(Iter {iteration})")
        axes[i, 0].axis('off')

        # Noisy Image - Squeeze the channel dimension (1) to make it (H, W)
        axes[i, 1].imshow(noisy_img_np.squeeze(), cmap='gray')
        axes[i, 1].set_title(f"Noisy (PSNR: {compare_psnr(gt_img_np.squeeze(),
                                            noisy_img_np.squeeze()):.2f})")
        axes[i, 1].axis('off')

        # DIP Estimated Image - Squeeze the channel dimension (1) to make it (H, W)
        estimated_data = visual_data.get(iteration)
        if estimated_data:
            axes[i, 2].imshow(estimated_data['out_np'].squeeze(), cmap='gray')
            axes[i, 2].set_title(f"DIP ({loss_type})\nPSNR: {estimated_data['psnr']:.2f}
                                            ")
        else:
            axes[i, 2].set_title(f"DIP ({loss_type})\nData missing for {iteration}")
        axes[i, 2].axis('off')

    plt.suptitle(f"Denoising Examples for {image_title} ({loss_type} Loss)", y=1.02,
                                            fontsize=16)
    plt.tight_layout(rect=[0, 0.03, 1, 0.98])
    plt.savefig(f"denoising_examples_{os.path.splitext(image_title)[0]}_{loss_type.lower
                                            ()}.png")
    plt.show()


# Prepare data for visualization and call plotting function
print(f"\n--- Preparing Visual Examples for {VISUAL_EXAMPLES_IMAGE_FNAME} ({
                                            current_loss_type_name} Loss) ---")
# Load GT and Noisy image for the selected example only once for plotting
gt_img_pil = crop_image(get_image(os.path.join('../test_Set/',
                                            VISUAL_EXAMPLES_IMAGE_FNAME), imsize)[0].
                                            convert('L'),
                        d=32)
gt_img_np = pil_to_np(gt_img_pil)

# Re-add noise just for the noisy image display, ensuring it's consistent for the plot
np.random.seed(0)
temp_noise = np.random.normal(0, sigma, gt_img_np.shape).astype(np.float32)
noisy_img_for_display_np = np.clip(gt_img_np + temp_noise, 0, 1)

plot_visual_examples(gt_img_np, noisy_img_for_display_np, visual_data_for_selected_image
                                            ,
                     dynamic_visual_iterations_global, VISUAL_EXAMPLES_IMAGE_FNAME,
                                            current_loss_type_name)
```