Practical Work 1: Basics of Image Processing and Denoising

Install

i. Open 'FCMR40' virtual machine using VirtualBox. Password: fcmr40

ii. A little update:

   `sudo sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list`

   `sudo apt-get update && sudo apt-get dist-upgrade`

   `sudo apt install virtualenv`

iii. Get the code repository

   Open a terminal: Ctrl + Alt + T

   `cd ~/Desktop`

   `git clone https://github.com/FlorianLemarchand/formation_eavesdropping_denoising.git`

iv. Setup the python environment

   `cd formation_eavesdropping_denoising`

   - Create virtual environement:

   `virtualenv -p python3 venv`

   - Connect to the virtual environement:

   `source venv/bin/activate`

   The command line should now look like:

   `(venv) toto:~$`

   From now, each python package will be installed in this virtual environement. To exit the virtual environement:

   `deactivate`

v. Install the required packages:

   `pip3 install -r requirements_pw1.txt`

   - You can display the installed packages and their versions using:

   `pip3 list`

1. Image Manipulations

In this section, you will use some basics code lines to manipulate images. For simplicity, write the code in the python script 'pw1.py'. You can run this script from the terminal:

`python3 pw1.py`

i. Load images 'im1.jpg' and 'im2.jpg' from 'data' directory.

```
im1 = imread('data/in/im1.jpg')
```

ii. Print their shapes, means, standard deviation (std), mins ans maxs:

```
# Print the shapes, means, std, mins, maxs of the images
print('Im1 ==> Shape: {} / Mean: {} / Std: {} / Min: {} / Max: {}'.format(im1.shape,
                                                                          np.round(np.mean(im1), 2),
                                                                          np.round(np.std(im1), 2),
                                                                          np.min(im1),
                                                                          np.max(im1)))

print('Im2 ==> Shape: {} / Mean: {} / Std: {} / Min: {} / Max: {}'.format(im1.shape,
                                                                          np.round(np.mean(im2), 2),
                                                                          np.round(np.std(im2), 2),
                                                                          np.min(im2),
                                                                          np.max(im2)))
```

   Which dimensions are the height, width and number of channels? Note that this order is not the same for all libraries!

iii. Save/Display the images:

```
# Save the patches
makedirs('data/out', exist_ok=True)
imsave('data/out/crop_1.jpg', crop_1)
imsave('data/out/crop_2.jpg', crop_2)

# Display the images
plt.subplot(2,1,1)
plt.title('Image 1')
plt.imshow(im1)
plt.subplot(2,1,2)
plt.title('Image 2')
plt.imshow(im2)
plt.show()
```

iv. Play with Python indexing to manipulate images and display them:

```
# Crop patchs
crop_1 = im1[0:150, 0:150, :]
crop_2 = im2[0:100, 150:250, :]

# Flip the image
vflip = im1[::-1, :, :]
hflip = im1[:, ::-1, :]

# Rotate the image
rota90 = vflip.transpose(1, 0, 2)
rota180 = im1[::-1, ::-1, :]
rota270 = hflip.transpose(1, 0, 2)

# Down-Sample the image
stride4 = im1[0:-1:4, 0:-1:4, :]
stride8 = im1[0:-1:8, 0:-1:8, :]
```

v. Compute horizontal and vertical gradients

```
# Transform to grayscale
im1_gray = rgb2gray(im1)
print('Grayscale image shape:', im1_gray.shape)

# Compute horizontal and vertical gradients
vgrad = im1_gray[0:-2, :] - im1_gray[1:-1, :]
hgrad = im1_gray[:, 0:-2] - im1_gray[:, 1:-1]
```

2. Synthetic Noising and Quality Assessement

In this section you will experience synthetic image noising and quality assessment. For that end the scikit-image python package is used.

i. Generate a map of white Gaussian noise with standard deviation 50. Display its distribution through an histogram. Add the noise to im1.

```
sigma = 50
# Cast the image and parameters to float : each value coded on 64-bit float with value in [0,1]
im1_gray = img_as_float(im1_gray)
sigma_float = sigma/255.
variance = np.square(sigma_float) # variance = np.square(std)

# generate a noisy value for each pixel of im1_gray
noise_map = np.random.normal(0., sigma_float,  im1_gray.shape)

# Add noise to image
im_noise = im1_gray + noise_map

# Display the value histogram of the noise map, the original and noisy images
vec_noise = np.reshape(noise_map, noise_map.size)  # Vectorize noise_map
plt.subplot(2,2,1)
plt.imshow(im1_gray, cmap='gray')
plt.subplot(2,2,2)
plt.hist(vec_noise, bins=1000)
plt.subplot(2,2,3)
plt.imshow(im_noise, cmap='gray')
plt.show()
```

ii. Use a library to do the same noising in one line:

```
im_noise_lib = random_noise(im1_gray, 'gaussian', mean=0., seed=0, var=variance, clip=False)
```

- *Display 'im_noise_lib' and its histogram.*

iii. Quality assessment of the noisy image:

```
# Measure the quality of the noisy images with respect to the clean image
psnr = compare_psnr(im_noise, im1_gray)
mse = compare_mse(im_noise, im1_gray)
ssim = compare_ssim(im_noise, im1_gray)
print('im_noise: PSNR: {} / MSE: {} / SSIM: {}'.format(psnr, mse, ssim))
```

iv. Display different intensities of noise applied to im1:

```
# Noise im1 with different sigmas
sigmas = range(20, 161, 20)
ncol = 3
nrow = int((len(sigmas) + 2) / ncol)

print('{} image to display on {} columns and {} rows'.format(len(sigmas) + 1, ncol, nrow))

plt.subplot(nrow, ncol, 1)
psnr = np.round(compare_psnr(im1_gray, im1_gray), 2)
ssim = np.round(compare_ssim(im1_gray, im1_gray), 2)
plt.title('Sigma: {} \nPSNR:{}/SSIM:{}'.format(0, psnr, ssim))
plt.imshow(im1_gray, cmap='gray')
for i, sigma in enumerate(sigmas):
    var = np.square(sigma / 255.)  # Compute variance from sigma
    im_noise = random_noise(im1_gray, 'gaussian', mean=0, var=var, seed=0, clip=True)  # Noise the image
    psnr = np.round(compare_psnr(im_noise, im1_gray), 2)  # Compute PNSR
    ssim = np.round(compare_ssim(im_noise, im1_gray), 2)  # Compute SSIM
    plt.subplot(nrow, ncol, i + 2)
    plt.title('Sigma: {} \n{}/{}'.format(sigma, psnr, ssim))
    plt.imshow(im_noise, cmap='gray')

plt.show()
```

v. Try other noise types in the 'random_noise' function (Documentation). Measure the resulting PSNR and SSIM.

vi. Apply sequentially two different noise types and measure the metrics.

3. Denoising using basic filtering

In this section you will take a step towards denoising. Basic filtering will be used and the quality measured and observed to highlight the limits of such basic processings.

i. Filtering Framework

```
# Measure noisy PSNR/SSIM
print_psnr_ssim(im_noise_lib, im1_gray, 'Noisy')

# Filter loop function
def filter_loop(noisy_image, kernel, padding_type='zeros'):
height, width = noisy_image.shape
kernel_size = kernel.shape[0]

# Initialize the output array
output = np.zeros(noisy_image.shape)

# Generate padding
padding_size = int(kernel_size/2)

if padding_type is 'zeros':
    padded_input = np.zeros((height + 2 * padding_size, width + 2 * padding_size))
elif padding_type is 'ones':
    padded_input = np.ones((height + 2 * padding_size, width + 2 * padding_size))

padded_input[padding_size:padding_size+height, padding_size:padding_size+width] = noisy_image

# Loop over the image
for i in range(0, height):
    for j in range(0, width):
        output[i, j] = np.sum(np.multiply(kernel, padded_input[i:i+kernel_size, j:j+kernel_size]))

return output

# Use function to mean filter
hand_denoised = filter_loop(im_noise_lib, np.ones((3, 3)) / 9.)

# Measure denoised PSNR/SSIM
print_psnr_ssim(hand_denoised, im1_gray, 'Mean Denoised')
```

ii. Approximate Gaussian Filtering

```
  # Measure noisy PSNR/SSIM
  print_psnr_ssim(im_noise_lib, im1_gray, 'Noisy')
  # Filter
  def mean_filter(input, kernel_size):
kernel = np.ones((kernel_size, kernel_size))
output = 1 / np.square(kernel_size) * convolve(input, kernel)
return output
  mean_denoised = mean_filter(im_noise_lib, 3)

  # Measure denoised PSNR/SSIM
  print_psnr_ssim(gauss_denoised, im1_gray, 'Approximate Gaussian Denoised')
```

iii. Use scipy.ndimage library to try other filters: maximum_filter, minimum_filter, median_filter.

iv. Low Pass Transform Denoising

```
  # Transform the image and shift the result
  fft_clean = fftshift(fft2(im1_gray))
  fft_noisy = fftshift(fft2(im_noise_lib))

  shape = fft_noisy.shape
  middle_y, middle_x = int((shape[0] + 1.)/2.), int((shape[1]+1.)/2.)

  # Keep only a part of the coefficients
  percentage_keep = 0.2
  fft_thresh = fft_noisy.copy()  # init an output image
  fft_thresh[:, :] = 0.  # Set to 0. +0.j
  half_y_keep = int(((int(percentage_keep * shape[0]))+1)/2.)  # Compute the size of the window to keep
  half_x_keep = int(((int(percentage_keep * shape[1]))+1)/2.)
  fft_thresh[middle_y - half_y_keep: middle_y + half_y_keep, middle_x - half_x_keep: middle_x + half_x_keep] = \
  fft_noisy[middle_y - half_y_keep: middle_y + half_y_keep, middle_x - half_x_keep: middle_x + half_x_keep]

  # Inverse transform
  im_denoised = ifft2(ifftshift(fft_thresh)).real
  im_denoised = np.clip(im_denoised, 0., 1.)  # Ensure the values stay in [0., 1.]

  print_psnr_ssim(im_denoised, im1_gray, 'Denoised')

  # Display
if True:
    plt.subplot(3, 2, 1)
    plt.imshow(im1_gray, cmap='gray')
    plt.title('Clean Image')

  plt.subplot(3, 2, 2)
  plt.imshow(np.abs(fft_clean), norm=LogNorm())
  plt.title('Clean Spectrum')
  plt.colorbar()

  plt.subplot(3, 2, 3)
  plt.imshow(im_noise_lib, cmap='gray')
  plt.title('Noisy Image')

  plt.subplot(3, 2, 4)
  plt.imshow(np.abs(fft_noisy), norm=LogNorm())
  plt.title('Noisy Spectrum')
  plt.colorbar()

  plt.subplot(3, 2, 5)
  plt.imshow(im_denoised, cmap='gray')
  plt.title('Denoised Image')

  plt.subplot(3, 2, 6)
  plt.imshow(np.abs(fft_thresh), norm=LogNorm(vmin=0.1))
  plt.title('Denoised Spectrum')
  plt.colorbar()

  plt.show()
```

Do you think better denoising can be achieved? Let see in the next practical work what is the state of the art of denoising.