ECE 285 – Assignment #4
Non-local means

Written by Charles Deledalle on April 30, 2019.

In this assignment we will implement the non-local means algorithm as part of our image manipulation library `imagetools`.

First, start a Jupyter Notebook, go into the subdirectory `ece285_IVR_assignments` (or whatever you named it), and create a new notebook `assignment4_nlmeans.ipynb` with

```
%load_ext autoreload
%autoreload 2
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import time
import imagetools.assignment4 as im

%matplotlib notebook
```

We will use the following images:

- `assets/castle.png`
- `assets/leopard.png`

For the following questions, please write your code and answers directly in your notebook. Organize your notebook with headings, markdown and code cells (following the numbering of the questions).

1 Bilateral filter

The bilateral filter is a denoising algorithm that reads as:

$$x_{i,j}^{(\text{bilateral})} = \frac{1}{Z_{i,j}} \sum_{k=-s_1}^{s_1} \sum_{l=-s_2}^{s_2} \varphi\left(\frac{1}{C}\|y_{i+k,j+l} - y_{i,j}\|_2^2\right) y_{i+k,j+l} \quad \text{with} \quad Z_{i,j} = \sum_{k=-s_1}^{s_1} \sum_{l=-s_2}^{s_2} \varphi\left(\frac{1}{C}\|y_{i+k,j+l} - y_{i,j}\|_2^2\right) .$$

where φ is the so-called kernel function, C the number of channels (1 for grayscale, 3 for RGB), and $(-s_1, s_1) \times (-s_2, s_2)$ is the domain of the search window. We will choose $\varphi(\alpha) = \exp\left(-\frac{\max(\alpha - 2h\sigma^2, 0)}{2\sqrt{2}h\sigma^2/\sqrt{C}}\right)$, where σ^2 is the variance of the noise, assumed to be additive white and Gaussian, and $h > 0$ the filtering parameter.

1. From your notebook, load the image `x = castle` and add additive white Gaussian noise of standard deviation $\sigma = 10/255$ as

   ```
   sig = 10 / 255
   y   = x + sig * np.random.randn(*x.shape)
   ```

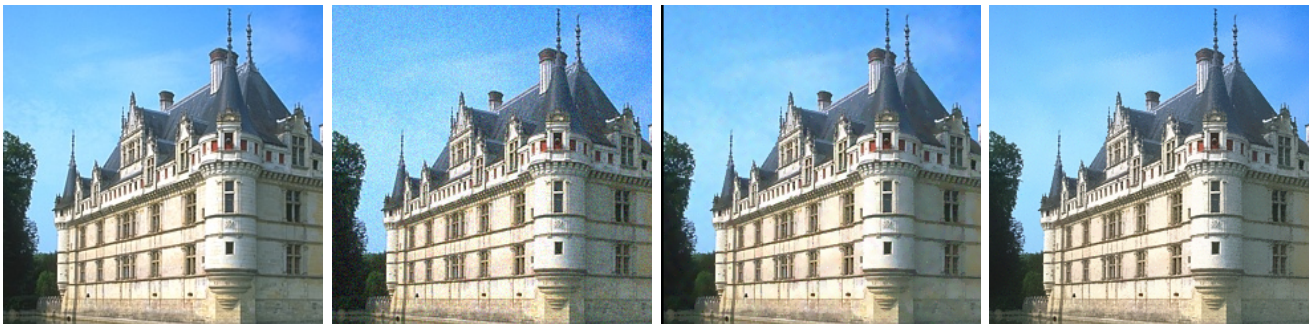
2. Create in `imagetools/assignment4.py`, a function

```
def bilateral_naive(y, sig, s1=2, s2=2, h=1):
    n1, n2 = y.shape[:2]
    c = y.shape[2] if y.ndim == 3 else 1
    ...
    return x
```

that implements the bilateral filter (except around boundaries). Note that the first and last lines are used to work likewise for grayscale images of shape `(n1, n2)`, and RGB images of shape `(n1, n2, 3)`. In this naive version use four loops as

```
x = np.zeros(y.shape)
Z = np.zeros((n1, n2, *[1] * (y.ndim - 2)))
for i in range(s1, n1-s1):
    for j in range(s2, n2-s2):
        for k in range(-s1, s1 + 1):
            for l in range(-s2, s2 + 1):
                dist2 = ((y[i + k, j + l] - y[i, j])**2).mean()
                # complete
Z[Z == 0] = 1
x = x / Z
```

3. Complete your notebook to test your function on `y` with $s_1 = s_2 = 2$ and $h = 1$. Zoom on the results to check that your functions are consistent with the following ones:



(a) Original (b) Noisy (c) Naive ($s = 2$, 88s) (d) Bilateral ($s = 7$, 2.9s)

4. Create in `imagetools/assignment4.py`, the function

```
def bilateral(y, sig, s1=10, s2=10, h=1, boundary='mirror')
```

that implements the bilateral filter including around boundaries. The idea is to switch the k, l loops with the i, j loops, and then make use of `shift`. The final code should read with only two loops and deal with boundary conditions.

5. Test your new function and compare the computation times with the naive approach.

6. Increase the search window size to $s_1 = s_2 = 7$. Increase the noise level σ. What do you observe?

2 NL-means filter

The NL-means filter is an extension of the bilateral filter, where instead of averaging values of pixels with similar values, the values of pixels centered on similar patches are averaged. The NL-means reads as:

$$x_{i,j}^{\text{(NL-means)}} = \frac{1}{Z_{i,j}} \sum_{k=-s_1}^{s_1} \sum_{l=-s_2}^{s_2} \varphi\left(\frac{1}{CP} \sum_{u=-p_1}^{p_1} \sum_{v=-p_2}^{p_2} \|y_{i+k+u,j+l+v} - y_{i+u,j+v}\|_2^2\right) y_{i+k,j+l} \tag{1}$$

$$\text{with} \quad Z_{i,j} = \sum_{k=-s_1}^{s_1} \sum_{l=-s_2}^{s_2} \varphi\left(\frac{1}{CP} \sum_{u=-p_1}^{p_1} \sum_{v=-p_2}^{p_2} \|(y_{i+k+u,j+l+v} - y_{i+u,j+v}\|_2^2\right) , \tag{2}$$

where $(-p_1, p_1) \times (-p_2, p_2)$ is the domain of a patch and $P = (2p_1 + 1) \times (2p_2 + 1)$ is its size. We will choose the following kernel function $\varphi(\alpha) = \exp\left(-\frac{\max(\alpha - 2h\sigma^2, 0)}{2\sqrt{2}h\sigma^2/\sqrt{CP}}\right)$.

7. Create in `imagetools/assignment4.py`, the function

```
def nlmeans_naive(y, sig, s1=2, s2=2, p1=1, p2=1, h=1)
```

that implements (except around boundaries) the NL-means filter with six loops as

```
for i in range(s1, n1-s1-p1):
    for j in range(s2, n2-s2-p2):
        for k in range(-s1, s1 + 1):
            for l in range(-s2, s2 + 1):
                dist2 = 0
                for u in range(-p1, p1 + 1):
                    for v in range(-p2, p2 + 1):
                        # complete
```

Note that your function should work likewise for grayscale images of shape `(n1, n2)`, or RGB images of shape `(n1, n2, 3)`.

8. In your notebook, load the image `x = leopard` and create `y` by adding additive white Gaussian noise of standard deviation $\sigma = 20/255$. Test your function on `y` with $s_1 = s_2 = 2$ and $p_1 = p_2 = 1$. Note: This question can take 10-15 mins to run.

9. Create in `imagetools/assignment4.py`, the function

```
def nlmeans(y, sig, s1=7, s2=7, p1=None, p2=None, h=1, boundary='mirror'):
    p1 = (1 if y.ndim == 3 else 2) if p1 is None else p1
    p2 = (1 if y.ndim == 3 else 2) if p2 is None else p2
    ...
```

that implements the NL-means filter (including around boundaries) with two loops only. What is the patch size?

Hint: you just need to use the function `convolve`.

10. In your notebook, test your new function with $s_1 = s_2 = 2$ and $p_1 = p_2 = 1$ and compare the results and check that your new function is more than 500 times faster.

11. What is the complexity of both approaches?

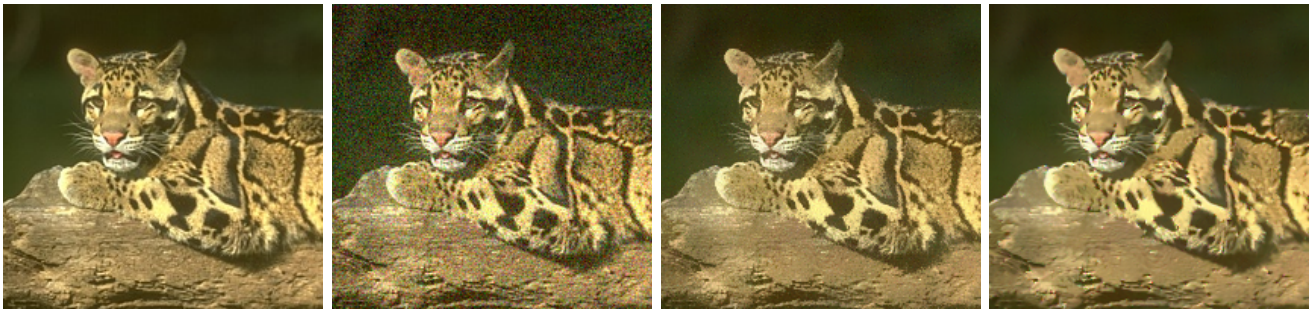
12. A very classical (but controversial) way to compare the quality of restoration techniques is to use the PSNR (Peak Signal-to-Noise-Ratio) defined for images as

$$\mathrm{PSNR} = 10 \log_{10} \frac{R^2}{\frac{1}{n}\|x - x_0\|_2^2} \tag{3}$$

where x_0 is the ideal image, x the estimate obtained from y, R the range of pixel values (255 for 8bits images, 1 for image in $[0, 1]$, etc), and n the number of elements in x. The PSNR measures in decibels (dB) the quality of the restoration: the higher the better. Implement the function:

```
def psnr(x, x0)
```

13. Increase to $s_1 = s_2 = 7$. Compare with the bilateral filter. Share axes and zoom on the results to check that your function is consistent with the following ones:



(e) Original (f) Noisy (g) Bilateral (29.82dB, 3s) (h) NLM (30.46dB, 5s)

14. Play with the noise level and the filtering parameter h. What do you observe?