ECE 285 – Assignment #2
Basic Image Tools

Written by Charles Deledalle on May 1, 2019.

In this assignment we will embellish our image manipulation package `imagetools` with very basic functions.

First, start a Jupyter Notebook, go into the subdirectory `ece285_IVR_assignments` (or whatever you named it), and create a new notebook `assignment2_basics.ipynb` and import

```
%load_ext autoreload
%autoreload 2
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import time
import imagetools.assignment2 as im

%matplotlib notebook
```

We will use the files

- `assets/lake.png`
- `assets/windmill.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 Image shift

Shifting an image x of size (n_1, n_2) in a direction (k, l) consists of creating a new image x^{shifted} of size (n_1, n_2) such that

$$x^{\text{shifted}}[i, j] = x[i + k, j + l] \ .$$

In practice, boundary conditions should be considered for pixels (i, j) such that $(i + k, j + l) \notin [0, n_1 - 1] \times [0, n_2 - 1]$. A typical example is to consider periodical boundary conditions such that

$$x^{\text{shifted}}[i, j] = x[i + k \mod n_1, j + l \mod n_2] \ .$$

1. Create in `imagetools/assignment2.py` a function implementing the shifting of an image x as

   ```
   def shift(x, k, l, boundary='periodical')
       ...
       return xshifted
   ```

   where the fourth argument is a string that specifies the type of boundary conditions to use (refer to the class). It takes one of the values: `'periodical'`, `'extension'`, `'zero-padding'` or `'mirror'`.

   Hint: First write it using loops as

```
n1, n2 = x.shape[:2]
if boundary is 'periodical_naive':
    xshifted = np.zeros(x.shape)
    # Main part
    for i in range(max(-k, 0), min(n1-k, n1)):
        for j in range(max(-l, 0), min(n2-l, n2)):
            xshifted[i, j] = x[i + k, j + l]
    # Corners
    for i in range(n1 - k, n1):
        for j in range(n2 - l, n2):
            xshifted[i, j] = x[i + k - n1, j + l - n2]
    for i in range(n1 - k, n1):
        for j in range(0, -l):
            xshifted[i, j] = x[i + k - n1, j + l + n2]
    for ...
```

Next try to get rid of the loops. Each case can be written with few lines, as:

```
if boundary is 'periodical':
    irange = np.mod(np.arange(n1) + k, n1)
    jrange = np.mod(np.arange(n2) + l, n2)
    xshifted = x[irange, :][:, jrange]
```

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

Note: if you cannot figure out all the cases, implement periodical and one more from the other three to receive full credits for this question, and then move to the next questions! Although, implementing all is highly recommended.

2. Complete your notebook to test your `shift` function for the boundary conditions implemented on `x = windmill`. The script should produce one figure with subplots as follows (do not forget to add a title to each subplot):

(a) Original (b) Periodical (c) Extension (d) Zero-padding (e) Mirror

Figure 1: Shift in the direction $(k, l) = (+100, -50)$ with different boundary conditions

3. Check on `x = windmill` and `y = lake`, if this operation is linear, i.e.,

```
shift(a * x + b * y, k, l) = a * shift(x, k, l) + b * shift(y, k, l) ?
```

for any arbitrary values of `a` and `b`. Does that depend on the boundary conditions?

Hint: use `np.allclose`.

4. After shifting the image in the direction (k, l), shift it back in the direction $(-k, -l)$. Interpret the results. Which shift is one-to-one? What is the null-space of this operation?

2 Image convolution

5. Create in `imagetools/assignment2.py`, the function

```
def kernel(name, tau=1, eps=1e-3):
    ...
    return nu
```

that produces a $(2s_1+1)\times(2s_2+1)$ array ν encoding a convolution kernel of finite support $(-s_1, s_1)\times(-s_2, s_2)$, defined as

$$\nu[i,j] = \frac{1}{Z} f(i - s_1, j - s_2) \quad \text{and} \quad Z = \sum_{i=-s_1}^{s_1} \sum_{i=-s_2}^{s_2} f(i,j) \tag{1}$$

where f is the kernel function to be specified and Z is the normalization constant. Note that the central value is $\nu[s_1, s_2]$. The first argument `name` is a string that specifies the type of kernel (refer to the class): `'gaussian'`, `'exponential'` or `'box'`. Because these functions f may have an infinite support, the values of s_1 and s_2 must be automatically determined as the smallest integers such that all locations (i, j) satisfying $f(i, j) > \varepsilon$ are located inside the window, *i.e.*, $-s_1 \leqslant i \leqslant s_1$ and $-s_2 \leqslant j \leqslant s_2$. Make sure that for the Gaussian kernel of bandwidth $\tau = 1$, you get $s_1 = s_2 = 3$, and for the exponential kernel of bandwidth $\tau = 3$, you get $s_1 = s_2 = 20$.

Hint: use `np.meshgrid(..., indexing='ij')`.

6. Create in `imagetools/assignment2.py`, the function

```
def convolve_naive(x, nu)
```

that performs (except around boundaries) the convolution between x and ν (with odd shape) with four loops as

```
n1, n2 = x.shape[:2]
s1 = int((nu.shape[0] - 1) / 2)
s2 = int((nu.shape[1] - 1) / 2)
xconv = np.zeros(x.shape)
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):
                # complete
```

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

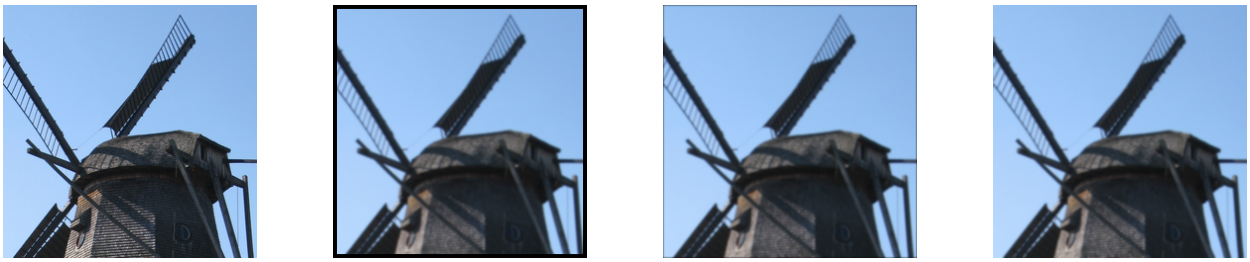
7. Create in `imagetools/assignment2.py`, the function

```
def convolve(x, nu, boundary='periodical')
```

that performs the convolution between x and ν 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 as

```
xconv = np.zeros(x.shape)
for k in range(-s1, s1+1):
    for l in range(-s2, s2+1):
        # complete
```

8. Compare the results and the execution times of `convolve_naive` and `convolve` for different boundary conditions (use `time.time()`). Check that you obtain something similar to



(a) Original (b) Naive (5.31s) (c) Spatial+ZP (0.03s) (d) Spatial+mirror (0.02s)

Figure 2: Gaussian blur with $\tau = 1$.

9. Check on `x = windmill` and `y = lake`, if this operation is linear. Does that depend on ν? on the boundary conditions?