ECE 285 – Assignment #3
Basic Filters

Written by Charles Deledalle on April 19, 2019.

In this assignment we will improve our image manipulation package `imagetools` by defining more advanced functions that we will be using for the next assignments.

First, start a Jupyter Notebook, go into the subdirectory `ece285_IVR_assignments` (or whatever you named it), and create a new notebook `assignment3_filters.ipynb` with

```
%load_ext autoreload
%autoreload 2
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import time
import imagetools.assignment3 as im

%matplotlib notebook
```

We will be using the files

- `assets/train.png`
- `assets/race.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 Separable convolutions

1. Copy paste the function `kernel` from `imagetools/assignment2.py` into `assignment3.py`. Modify the function in order to implement `name='gaussian1'`, `'exponential1'`, `'box1'`, `'gaussian2'`, `'exponential2'`, `'box2'`, that implements the Gaussian, exponential and box kernels but only in direction 1 and 2 respectively.

   Hint: just modify the value of `s1` and `s2` and use `name.startswith` and `name.endswith`.

2. Copy paste the function `convolve` from `imagetools/assignment2.py` into `assignment3.py`. Modify the function in order to implement cases where the kernel is separable. The function signature will be modified as follows

   ```
   def convolve(x, nu, boundary='periodical', separable=None)
   ```

   Here `separable` can take one of the values: `None`, `'product'` or `'sum'`. When `separable = None`, the behavior of this function is unchanged. When `separable = 'product'`, `nu` is a list composed

of two 1d arrays, `nu = ( nu1, nu2 )`, and the function performs the convolution of x by ν defined as

$$\nu(i,j) = \nu_1(i)\nu_2(j).$$

When `separable = 'sum'`, ν is defined as

$$\nu(i,j) = 1_{j=0}\nu_1(i) + 1_{i=0}\nu_2(j).$$

where $1_{\text{condition}} = 1$ if the condition is satisfied, 0 otherwise. Example: the following code

```
nu1   = im.kernel('gaussian1', tau)
nu2   = im.kernel('gaussian2', tau)
nu    = ( nu1, nu2 )
xconv = im.convolve(x, nu, boundary='mirror', separable='product')
```

should produce the same result as

```
nu    = im.kernel('gaussian', tau)
xconv = im.convolve(x, nu, boundary='mirror')
```

3. Perform the convolution of the image `x = train` with the Gaussian, box and exponential kernel. Compare the similarity of the results and the execution times of the separable version and the non-separable one. Display the results and check that your results are consistent with the followings:

(a) Original x (b) x^{conv} non-separable (2.10s) (c) x^{conv} separable (0.32s)

Figure 1: Results of convolutions with the exponential kernel $\tau = 4$.

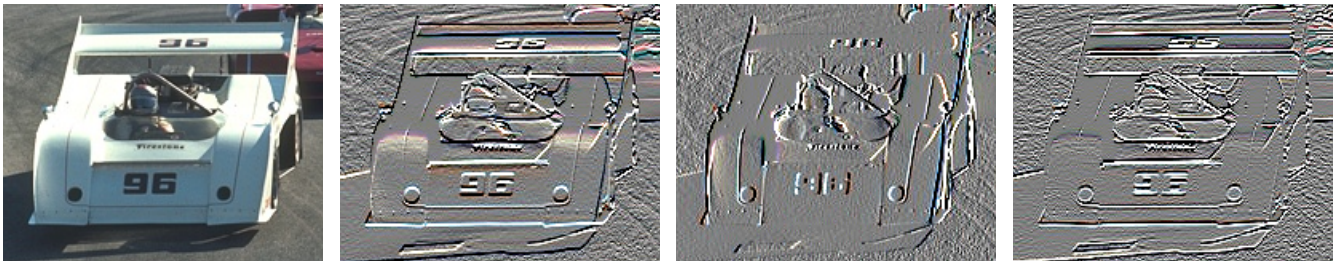
2 Derivative filters

4. Modify the function `kernel` such that it implements the following derivative filters: `grad1_forward` (forward discrete gradient in the first dimension), `grad1_backward`, `grad2_forward`, `grad2_backward`, `laplacian1` (Laplacian in the first dimension), `laplacian2` (Laplacian in the second dimension). Please refer to the class for proper definitions. For instance, the code for the forward discrete gradient in the first dimension reads as:

```
if name is 'grad1_forward':
    nu = np.zeros((3, 1))
    nu[1, 0] = -1
    nu[2, 0] = 1
```

In this case, the arguments `tau` and `eps` are ignored. A kernel should always have odd shape dimensions.

5. Apply the 6 derivative filters on `y = race` and display the 6 results (in the range [-.2, .2]). Share axes, zoom and move on the images, and check that your results are consistent with Figure 2.



(a) Original x (b) $\nabla_1^{\text{forward}} x$ (c) $\nabla_2^{\text{backward}} x$ (d) $\Delta_1 x$

Figure 2: Results of derivative filters.

6. For a 1d signal of size 4, the gradient with forward finite difference and periodical boundary condition can be expressed in matrix form as

$$\nabla_1^{\text{forward}} = \begin{pmatrix} -1 & 1 & & \\ & -1 & 1 & \\ & & -1 & 1 \\ 1 & & & -1 \end{pmatrix}$$

Derive the expression of the gradient with backward finite difference and periodical boundary condition. Show that $(\nabla_1^{\text{forward}})^T = -\nabla_1^{\text{backward}}$.

7. Two linear functions f and g are adjoint of each other (generalization of matrix transposition), if for all x and y, $\langle x, f(y)\rangle = \langle y, g(x)\rangle$. We note $g = f^*$. For `x = train` and `y = race`, check whether $\langle x, \nabla_1^{\text{backward}} y\rangle = -\langle \nabla_1^{\text{forward}} x, y\rangle$ up to machine precision (use `np.isclose`) for different boundary conditions. Conclude.

Reminder: $\langle x, y\rangle = \sum_k x_k y_k$.

8. Prove your observations from the previous questions using their corresponding matrix forms on 1d signals of size 4.

9. Check whether $\nabla_1^{\text{backward}} \nabla_1^{\text{forward}} y = \Delta_1 y$ up to machine precision for the different boundary conditions. What do you conclude?

10. Prove your previous observation on periodical boundary conditions, by using the corresponding matrix forms on 1d signals of size 4.

11. Create in `imagetools/assignment3.py`, the function

```
def laplacian(x, boundary='periodical')
```

that returns the Laplacian of the image `x`.

Hint: use `separable = 'sum'`.

12. Create the function

```
def grad(x, boundary='periodical')
    ...
    return g
```

that for a $n_1 \times n_2$ image `x` (resp., $n_1 \times n_2 \times 3$ for an RGB image) returns a $n_1 \times n_2 \times 2$ (resp., $n_1 \times n_2 \times 2 \times 3$) array `g` corresponding to the discrete gradient vector field of `x` with periodical boundary conditions. More precisely, `g[:,:,0]` (resp., `g[:,:,1]`) corresponds to the forward discrete image gradient in the first (resp., second) direction.

Hint: use `np.stack`.

13. The divergence of a two-dimensional vector field $f : \mathbb{R}^2 \to \mathbb{R}^2$ is defined as the function $\operatorname{div} f : \mathbb{R}^2 \to \mathbb{R}$ such that

$$\operatorname{div} f = \frac{\partial f_1}{\partial s_1} + \frac{\partial f_2}{\partial s_2}.$$

Create in `imagetools/assignment3.py`, the function

```
def div(f, boundary='periodical')
    ...
    return d
```

that for a $n_1 \times n_2 \times 2$ vector field `f` returns its $n_1 \times n_2$ discrete divergence `d`.

14. Check that your implementation guarantees that for periodical boundary conditions, $\operatorname{div} \nabla x = \Delta x$ and compare $\langle \nabla x, \nabla y \rangle = -\langle \Delta x, y \rangle$. Revise your code if it does not. Conclude.

Note: These properties will be essential for the algorithms we will develop next.