Examples

October 20, 2018

```
In [25]: import sys
         %matplotlib inline
         import matplotlib.pyplot as plt
         import numpy as np
         sys.path.append("src")
         import util.data.bmp as bmp
         from transform import normalize

In [2]: img=bmp.read_image("test/images/avion.bmp")

In [3]: def proccess_image(image, func, *args, **kwargs):
            result = dict()
            for chanel in image:
                result[chanel] = func(image[chanel],*args, **kwargs)
            return bmp._merge_rgb(result).astype(np.uint8)

        def from_dict(img):
            return bmp._merge_rgb(img).astype(np.uint8)

        def show_result(img, result):
            f, axarr = plt.subplots(1,2, figsize = (10, 15))
            axarr[0].imshow(from_dict(img))
            axarr[1].imshow(result)
```

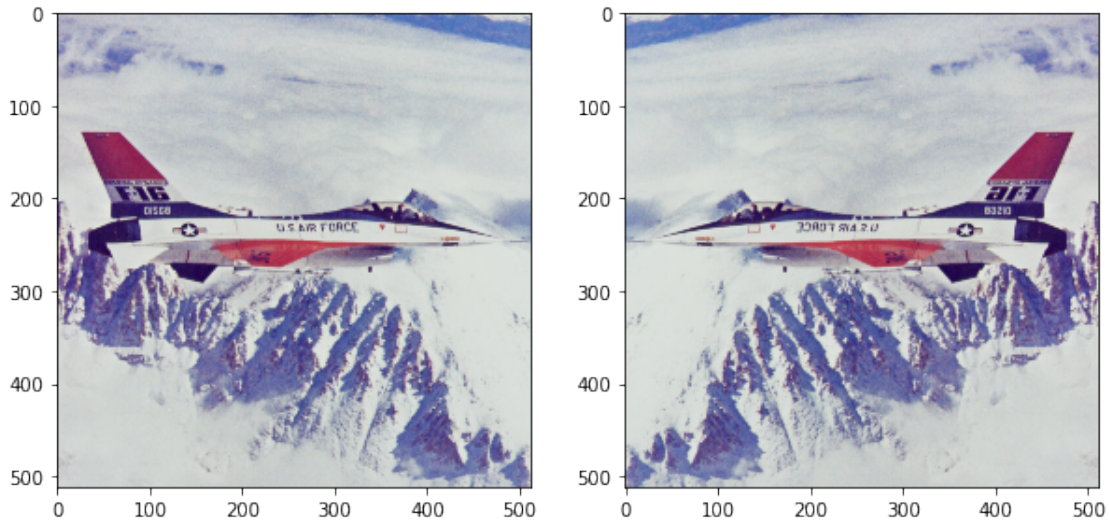
1

```
In [5]: from transform.rotate import rotate

In [21]: %timeit result = proccess_image(img, rotate, degree=90)

238 ms ś 1.14 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)


In [22]: show_result(img, result)
```



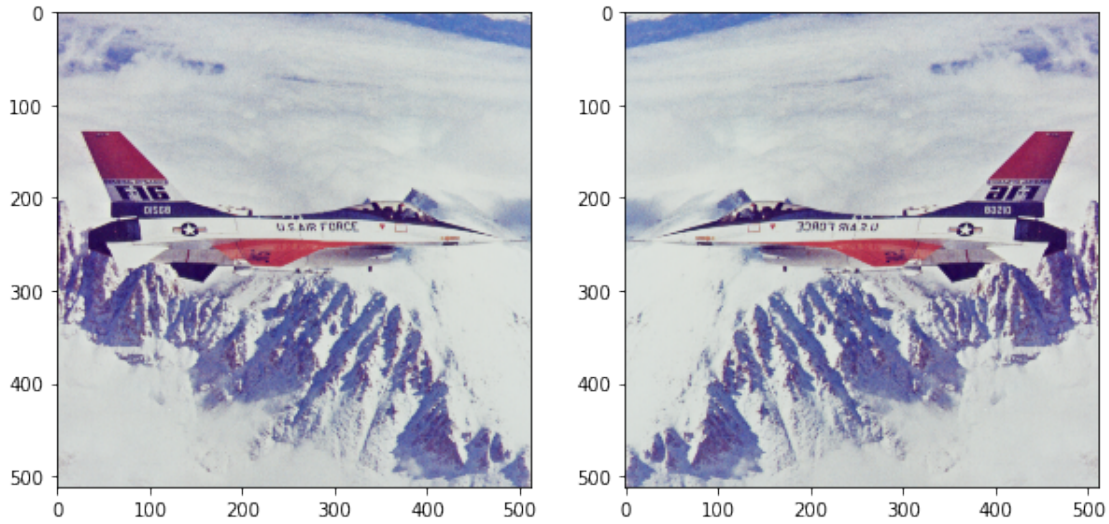
2

```
In [16]: from transform.mirror import mirror
```

```
In [23]: %timeit result = proccess_image(img, mirror, 'x')
```

```
117 ms ś 1.4 ms per loop (mean ś std. dev. of 7 runs, 10 loops each)
```

```
In [24]: show_result(img, result)
```



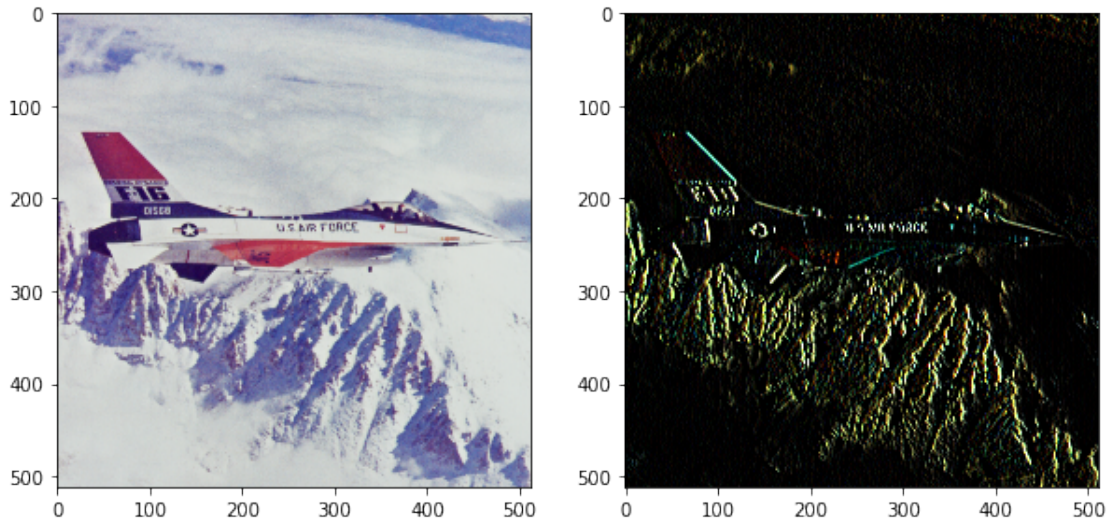
3

```
In [27]: from filter.sobel import sobel
         def add_128_sobel(*argc, **kwargs):
             res = sobel(*argc, **kwargs)
             fres = normalize.shift(res, 128)
             res = normalize.suppress_ejection(res)
             return res
```

```
In [29]: %timeit result = proccess_image(img, add_128_sobel, direction = 'x', mode = 'odd')
```

```
14.1 s ś 27.3 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)
```

```
In [32]: show_result(img, result)
```



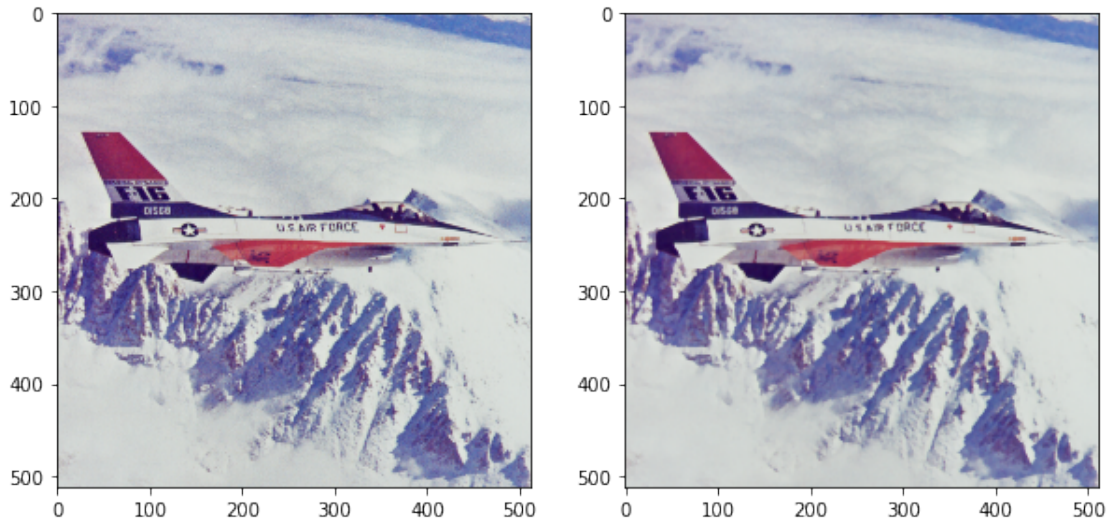
4

```
In [33]: from filter.median import median
```

```
In [34]: %timeit result = proccess_image(img, median, rad=1)
```

```
49.2 s ś 728 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)
```

```
In [35]: result = proccess_image(img, median, rad=1)
         show_result(img, result)
```



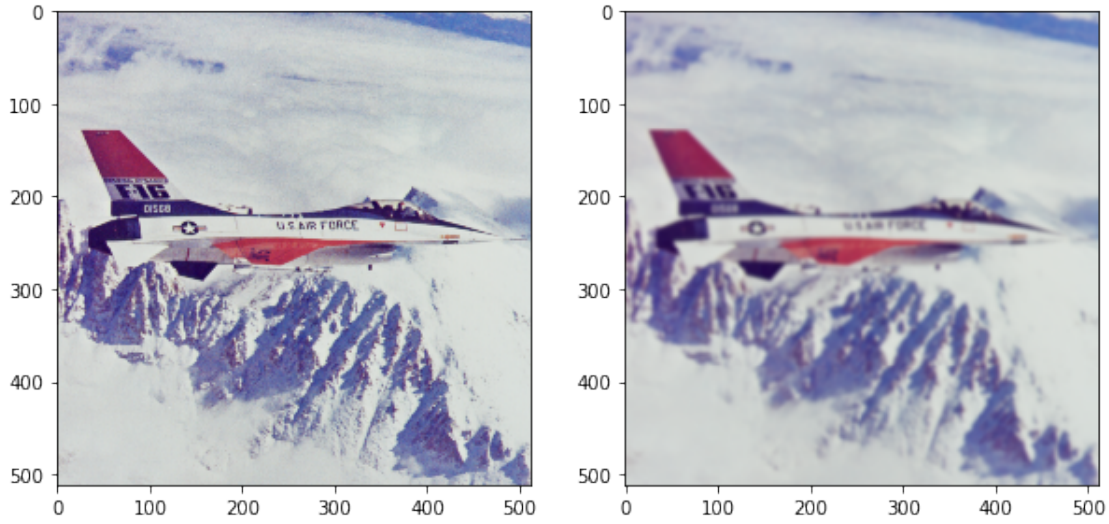
5

```
In [36]: from filter.gauss import gauss
```

```
In [39]: %timeit result = proccess_image(img, gauss, mode = 'rep', sigma = 2)
```

```
1min 9s ś 780 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)
```

```
In [40]: result = proccess_image(img, gauss, mode = 'rep', sigma = 2)
         show_result(img, result)
```



6

```
In [41]: from filter.gradient import gradient
         def normalized_gradient(*argc, **kwargs):
             res = gradient(*argc, **kwargs)
             res = normalize.suppress_ejection(res)
             return res
```

```
In [46]: %timeit result = proccess_image(img, normalized_gradient, 'rep', 1)
```

```
48.9 s ś 1.08 s per loop (mean ś std. dev. of 7 runs, 1 loop each)
```

```
In [42]: result = proccess_image(img, normalized_gradient, 'rep', 1)
         show_result(img, result)
```