benchmarking

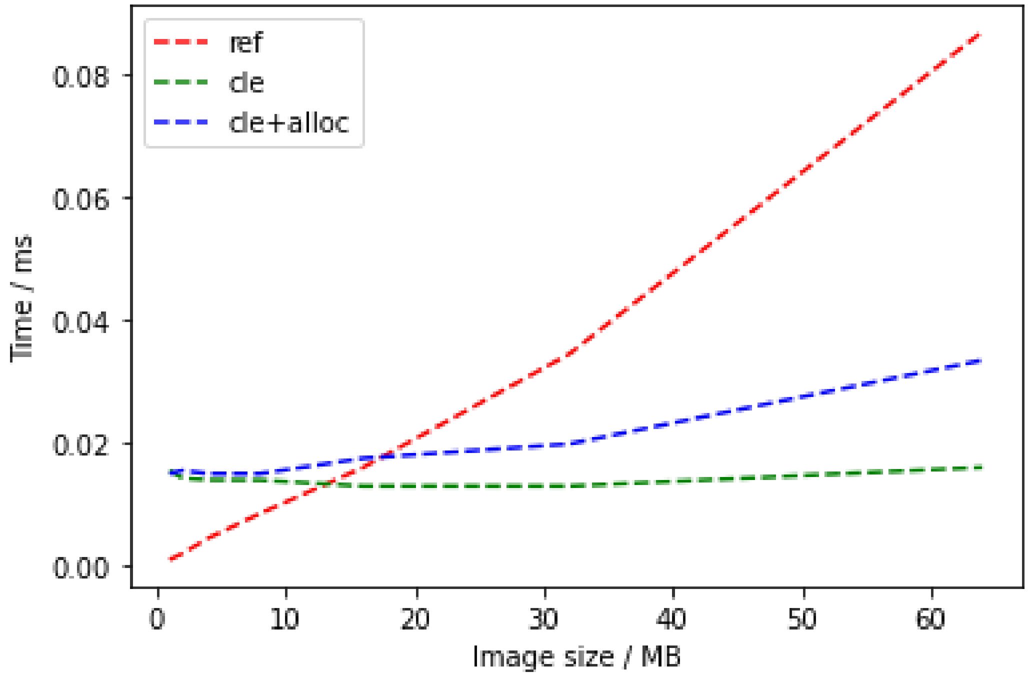
June 4, 2020

1 Benchmarking numpy / scikit-image / scipy vs clesperanto

```
In [1]: import clesperanto as cle
        import numpy as np
        import time
        import matplotlib.pyplot as plt

        num_iterations = 10

        # measure execution time of a given method
        def benchmark(function, kwargs):
            times = []
            for i in range(0, num_iterations):
                start_time = time.time()
                function(**kwargs)
                delta_time = time.time() - start_time
                times = times + [delta_time]
                # print(delta_time)

            # return median of measurements to ignore warmup-effects
            return np.median(times)



        def benchmark_size(method_np, method_cle, method_cle_alloc):
            times_ref = []
            times_cle = []
            times_cle_alloc = []
            sizes = []
            for size in [1, 2, 4, 8, 16, 32, 64]:

                input1 = np.zeros((1024, 1024, size))
                cl_input1 = cle.push(input1)
                cl_input2 = cle.create(cl_input1.shape)

                time_ref = benchmark(method_np, {"image":input1})
                time_cle = benchmark(method_cle, {"image":cl_input1, "output":cl_input2})
                time_cle_alloc = benchmark(method_cle_alloc, {"image":cl_input1})

                times_ref = times_ref + [time_ref]
                times_cle = times_cle + [time_cle]
                times_cle_alloc = times_cle_alloc + [time_cle_alloc]
                sizes = sizes + [size]

            plt.plot(sizes, times_ref,  'r--', sizes, times_cle, 'g--', sizes, times_cle_alloc
            plt.ylabel('Time / ms')
            plt.xlabel('Image size / MB')
            plt.legend(("ref", "cle", "cle+alloc"));
            plt.show()


            print("\nSizes (MB)        " + str(sizes))
            print("Times ref (s)       " + str(np.round(times_ref, 4)))
            print("Times cle (s)       " + str(np.round(times_cle, 4)))
            print("Times cle+alloc (s) " + str(np.round(times_cle_alloc, 4)))
```

1.1 Thresholding

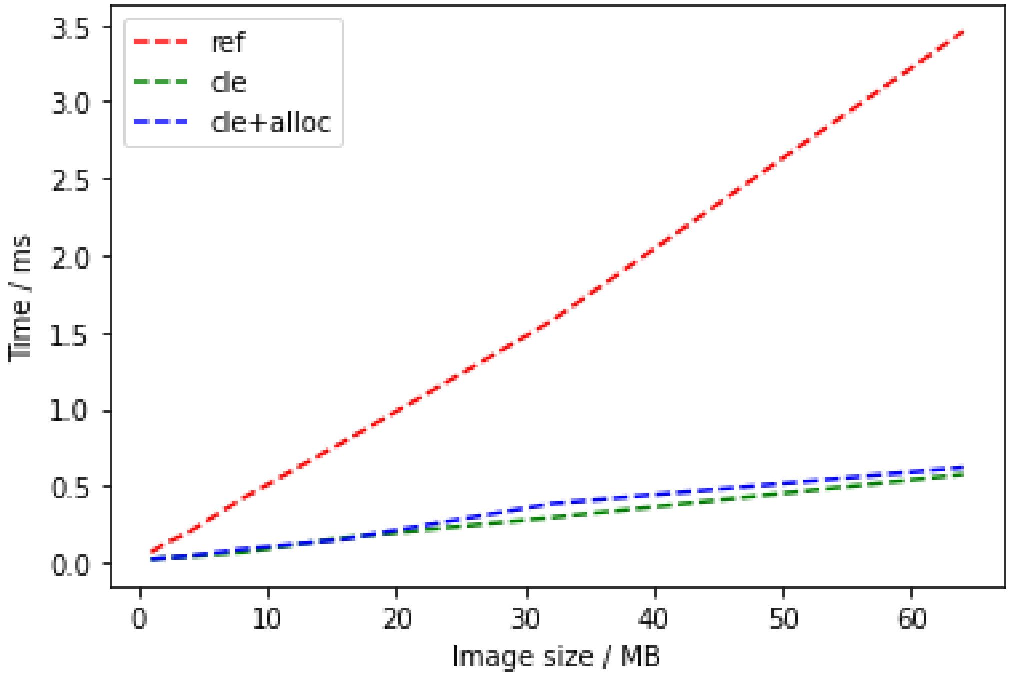
```
In [2]: # RED: thresholding of a numpy array
        def threshold_ref(image):
            thresholded = image > 100
            return thresholded

        # GREEN: thresholding of a pre-existing opencl array (no push, pull or alloc)
        def threshold_cle(image, output):
            cle.greater_constant(image, output, 100)

        # BLUE: allocate result memory + thresholding
        def threshold_cle_alloc(image):
            thresholded = cle.create(image.shape)
            cle.greater_constant(image, thresholded, 100)

        benchmark_size(threshold_ref, threshold_cle, threshold_cle_alloc)
```



```
Sizes (MB)         [1, 2, 4, 8, 16, 32, 64]
Times ref (s)        [0.001  0.002  0.0045 0.0085 0.016  0.0345 0.0868]
Times cle (s)        [0.0155 0.0143 0.014  0.0139 0.013  0.013  0.016 ]
Times cle+alloc (s) [0.015  0.0155 0.015  0.015  0.0175 0.0198 0.0334]
```

1.2 Gaussian blur radius 2

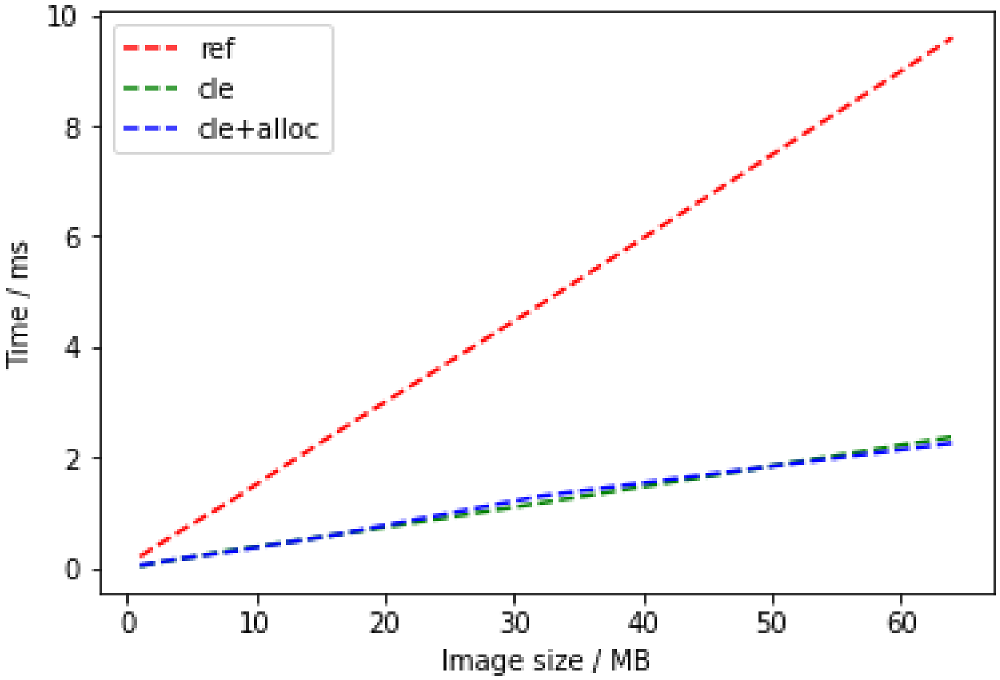
```
In [3]: from skimage.filters import gaussian

        radius = 2

        def gaussian_blur_filter_ref(image):
            filtered = gaussian(image, sigma=radius)
            return filtered

        def gaussian_blur_filter_cle(image, output):
            cle.gaussian_blur(image, output, radius, radius, radius)

        def gaussian_blur_filter_cle_alloc(image):
            filtered = cle.create(image.shape)
            cle.gaussian_blur(image, filtered, radius, radius, radius)

        benchmark_size(gaussian_blur_filter_ref, gaussian_blur_filter_cle, gaussian_blur_filte
```



```
Sizes (MB)         [1, 2, 4, 8, 16, 32, 64]
Times ref (s)        [0.0653 0.1192 0.1999 0.4114 0.7856 1.5673 3.4555]
Times cle (s)        [0.0199 0.0294 0.0399 0.0648 0.1616 0.292  0.5732]
Times cle+alloc (s) [0.0214 0.0289 0.0424 0.0853 0.1521 0.3822 0.6174]
```

1.3 Gaussian blur radius 10

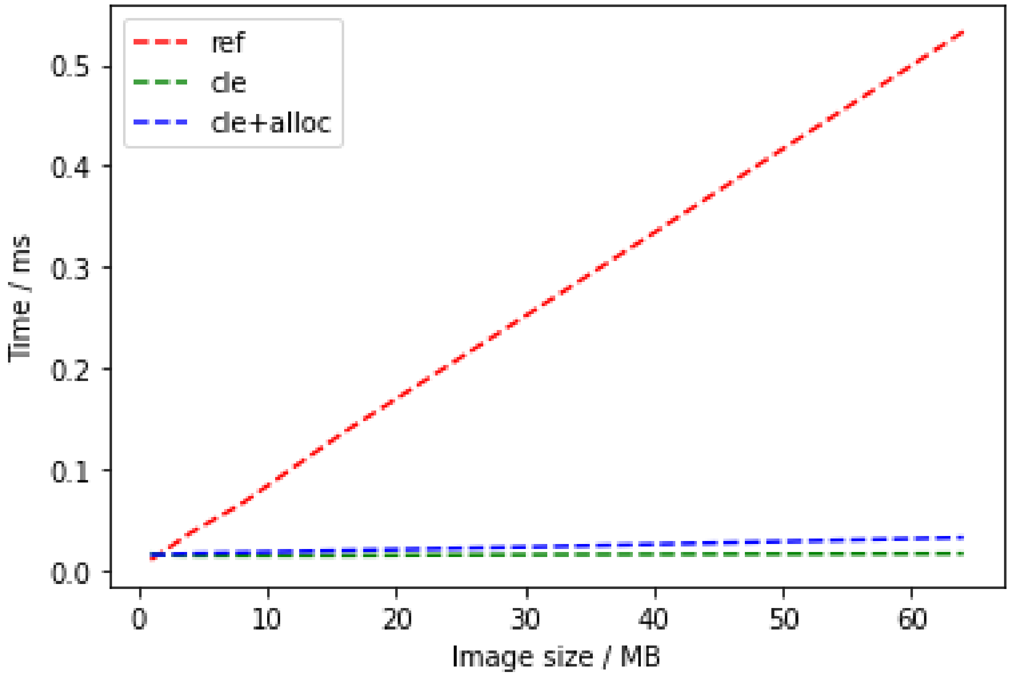
```
In [4]: radius = 10
        benchmark_size(gaussian_blur_filter_ref, gaussian_blur_filter_cle, gaussian_blur_filte
```



```
Sizes (MB)          [1, 2, 4, 8, 16, 32, 64]
Times ref (s)         [0.2077 0.3602 0.6532 1.2087 2.4212 4.7539 9.585 ]
Times cle (s)         [0.0399 0.0851 0.162  0.3166 0.5969 1.1758 2.3712]
Times cle+alloc (s) [0.0524 0.0905 0.168  0.3005 0.5974 1.3004 2.2642]
```

1.4 Binary erosion

```
In [5]: from skimage.morphology import binary_erosion


        def binary_erosion_ref(image):
            filtered = binary_erosion(image)
            return filtered

        def binary_erosion_cle(image, output):
            cle.erode_box(image, output)

        def binary_erosion_cle_alloc(image):
            filtered = cle.create(image.shape)
            cle.erode_box(image, filtered)

        benchmark_size(binary_erosion_ref, binary_erosion_cle, binary_erosion_cle_alloc)
```

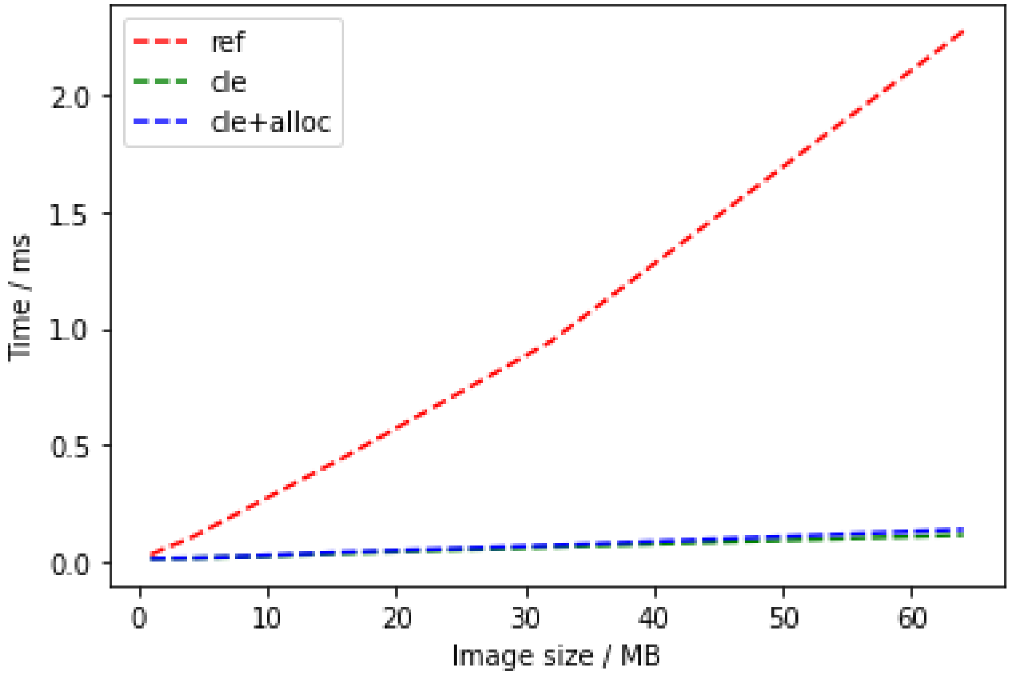


```
Sizes (MB)         [1, 2, 4, 8, 16, 32, 64]
Times ref (s)        [0.01   0.018  0.0359 0.0648 0.1362 0.2673 0.5328]
Times cle (s)        [0.016  0.0145 0.014  0.014  0.014  0.015  0.016 ]
Times cle+alloc (s) [0.015  0.015  0.016  0.017  0.019  0.0229 0.0319]
```

1.5 Mean filter radius=2

```
In [6]: import scipy.ndimage.filters as spf


        radius = 2
        def mean_filter_ref(image):
            # todo: not sure if size is a radius or a diameter. Check documentation
            # https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.uniform_filte
            filtered = spf.uniform_filter(image, size=radius)
            return filtered

        def mean_filter_cle(image, output):
            cle.mean_box(image, output, radius, radius, radius)

        def mean_filter_cle_alloc(image):
            filtered = cle.create(image.shape)
            cle.mean_box(image, filtered, radius, radius, radius)
```

```
benchmark_size(mean_filter_ref, mean_filter_cle, mean_filter_cle_alloc)
```



```
Sizes (MB)         [1, 2, 4, 8, 16, 32, 64]
Times ref (s)        [0.0339 0.0593 0.1032 0.2174 0.4493 0.9431 2.2677]
Times cle (s)        [0.017  0.018  0.0194 0.0249 0.0404 0.067  0.1185]
Times cle+alloc (s) [0.0175 0.0189 0.0199 0.0269 0.0455 0.0738 0.1417]
```

```
In [ ]:
```