Analysing Microscopy Screens with EBImage

Oleg Sklyar and Wolfgang Huber
European Bioinformatics Institute - EMBL
Wellcome Trust Genome Campus
Cambridge CB10 1SD, England
osklyar@ebi.ac.uk

May 3, 2007

The R-package EBImage provides functionality to perform *image processing* and *image analysis* on large sets of images in a programmatic fashion using the R language.

We use the term *image analysis* to describe the extraction of numeric features (*image descriptors*) from images and image collections [Russ2002]. Image descriptors can then be used for statistical analysis, such as classification, clustering and hypothesis testing, using the resources of R and its contributed packages.

Image analysis is not an easy task, and the definition of image descriptors depends on the problem. Analysis algorithms need to be adapted correspondingly. We find it desirable to develop and optimize such algorithms in conjunction with the subsequent statistical analysis, rather than as separate tasks. This is one of our motivations for writing EBImage as an R package.

We use the term *image processing* for operations that turn images into images with the goals of enhancing, manipulating, sharpening, denoising or similar [Russ2002]. While some image processing is often needed as a preliminary step for image analysis, image processing is not the primary goal of the package. We focus on methods that do not require interactive user input, such as selecting image regions with a pointer etc. Whereas interactive methods can be very effective for small sets of images, they tend to have limited throughput and reproducibility.

EBImage uses the [ImageMagick] library API's to implement much of its functionality in image processing and input/output operations.

Cell-based assays

Advances in automated microscopy have made it possible to conduct large scale cell-based assays with image-type phenotypic readouts. In such an assay, cells are grown in the wells of a microtitre

plate (often a 96- or 384-well format is used) under a condition or stimulus of interest. Each well is treated with one of the reagents from the screening library and the cells' response is monitored, for which in many cases certain proteins of interest are antibody-stained or labeled with a GFP-tag [Carpenter2004, Wiemann2004, Moffat2006, Neumann2006].

The resulting imaging data can be in the form of two-dimensional (2D) still images, three-dimensional (3D) image stacks or image-based time courses. Such assays can be used to screen compound libraries for the effect of potential drugs on the cellular system of interest. Similarly, RNA interference (RNAi) libraries can be used to screen a set of genes (in many cases the whole genome) for the effect of their loss of function in a certain biological process [Boutros2004].

Importing and handling images

Images in EBImage are stored in objects of S4 class *Image* that extends the R *array* class. The class encapsulates the most important image properies like size, resolution and colour mode, which are accessible by means of *accessor methods*. It is recommended not to access slot data directly as their structure might change in the future releases and only use accessor methods. The `get` and `set` accessor methods for the *Image*class are: `colorMode`, `imageData`, `fileName`, `compression`, `resolution`, and the `get`-only `features`. The colour mode can be either `Grayscale` or `TrueColor` with the corresponding integer constants defined in the package. See help pages for the corresponding functions for more detail.

New images can be created with the standard R constructor `new`, or using the wrapper function `Image`. The following example code generates a 100x100 pixel grayscale image with black and white vertical stripes [1]:

```
> im <- Image(0.0, c(100,100), Grayscale)
> im[ c(1:20, 40:60, 80:100), , ] = 1
```

As mentioned above, for input/output operations EBImage uses ImageMagick API, which support reading and writing of more than 95 image formats including JPEG, TIFF and PNG. The package can read and write multi-page images (image stacks or 3D images) or process multiple files simultaneously. For example, the following code demonstrates how to read all PNG files in the working directory into a single object of class `Image`, convert them to grayscales and save the output as a single multi-page TIFF file:

```
> files <- dir(pattern=".png")
> im <- read.image(files, TrueColor)
> img <- channel(im, "gray")
> write.image(img, "single_multipage.tif")
```

The package also provides an interactive way to select and load (multiple) images using the `choose.image` function and an alternative way of specifying the file name to write as in the following example:

```
> img <- choose.image()
> fileName(img) <- "single_multipage.tif"
> write.image(img)
```

[1] All examples in this vignette are based on version 1.9.12 (RC) of the package. Older versions have similar functions, that can have slightly different names.

If the output format does not support multiple frames (stacks), ImageMagick will automatically extend the provided filename with counter suffixes `-0, -1, ...` placed either before or after the file extension depending on your installation of ImageMagick. Alternatively, one can specify the same number of files as image frames in call to `write.image` or `fileName`. File names supplied to `write.image` will take precedence over

Besides operations on local image files, anonymous HTTP and FTP protocols are supported. The package can read from both and it can write to FTP only. These protocols are supported internally by ImageMagick and do not use R connections.

Displaying images

EBImage implements the method `display` to show images on screen interactively. By default (and if the package was compiled with GTK+ support) EBImage uses the GTK+ implementation of this function. It allows to display as many images simultaneously as the user needs and its interface is easily expandable programmatically. The alternative implementation uses the ImageMagick API's `display` function. This implementation is only intended for systems where GTK+ interface is not available and can only show one image at a time. The `GTK+` interface is known to be working on all supported architectures, including MS Windows.

In some cases users may want to use the generic R `image` function, the package provides a simple wrapper, which keeps the image aspect ration and allows to specify which frame has to be displayed as in the following example:

```
> display(abc)  # 3 images, interactively
> image(abc, 2) # displays the 2nd frame
```

Image processing

The ImageMagick library provides a number of image processing routines, so called *filters*. Many of those are ported to R in EBImage. The missing ones can be added as opportunity arises. Additional image processing routines implementing distance map and watershed transforms and others have been added to the package to power our work on cell-based assays.

Filters are implemented as S4 methods for class *Image*. The return value is usually an object of class *Image* as well. One can divide all the filters into four categories: image enhancement, segmentation and transformation, and colour correction. Some examples are listed below.

channel enables colour mode and colour channel conversions

sharpen, umask generate sharpened versions of an image

gblur applies the Gaussian blur to an image softening sharp edges and noise

thresh segments a grayscale image into a binary black-and-white image by the adaptive thresholding algorithm

opening, closing sequentially use erosion and dilation operators to enhance edges of objects in binary images and to reduce noise

distmap performs a Euclidean distance transform of a binary image, also known as *distance map*. On a distance map, values of pixels indicate how far are they away from the nearest background

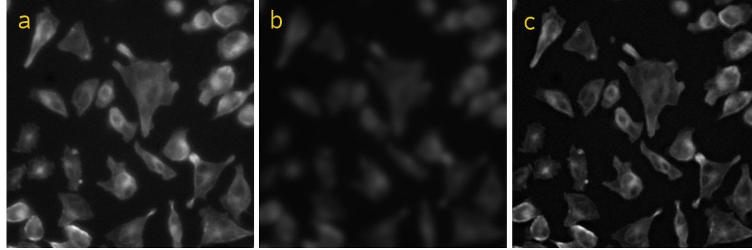

Figure 1: Implementation of a simple *unsharp mask* filter: (a) source image, (b) blurred colour-scaled image, (c) sharpened image after normalization

normalize shifts and scales colours of images to a specified range, normally $[0, 1]$ for grayscales

resample, resize changes image size, proportionally or arbitrary

watershed performs a *watershed*-like image segmentation (used also for object detection)

The storage mode of grayscale images is *double* (or *numeric*), and R functions that work with arrays can be directly applied to grayscale images. This includes the arithmetic functions, subsetting, histograms, Fast Fourier transformation, (local) regression etc. For example, the sharpened image in Figure 1**c** can be obtained by subtracting the slightly blurred, scaled in colour version of the original image (Figure 1**b**) from its source in Figure 1**a**. All pixels that become negative after subtraction are then re-set to background. The source image is obtained by subsetting the original microscopic image. Hereafter, variables in the code are given the same literal names as the corresponding image labels (e.g. data of variable `a` are shown in Figure 1 **a**, `b` – in **b**, and `C` – in **c**, etc).

```
> orig <- read.image("ch2.png")
> a <- orig[150:550, 120:520,]
> b <- blur(0.5 * a, 80, 5)
> C <- a - b
> C[C < 0] = 0
> C <- normalize(C)
```

One can think of this code as of a naive but fast and effective version of the *unsharp mask* filter; a more sophisticated implementation from the ImageMagick library is provided by the `umask` function in the package.

Some of the image analysis routines in EBImage assume grayscale data in the interval $[0, 1]$, but there are no formal restrictions on the range.

The storage mode of true colour images is *integer*, and we use the three lowest bytes to store the red (R), green (G) and blue (B) values, each in the `unsigned char` (of C) range of $[0, 255]$. Because of this, application of arithmetic and other functions on true colour images is generally meaningless; although they can be useful in some special cases, as shown in the example code in the following section. The support for true colour images is included mainly to enhance the display of the analysis results and to support further extensions of the package by third parties. Most analysis routines require grayscale data though.

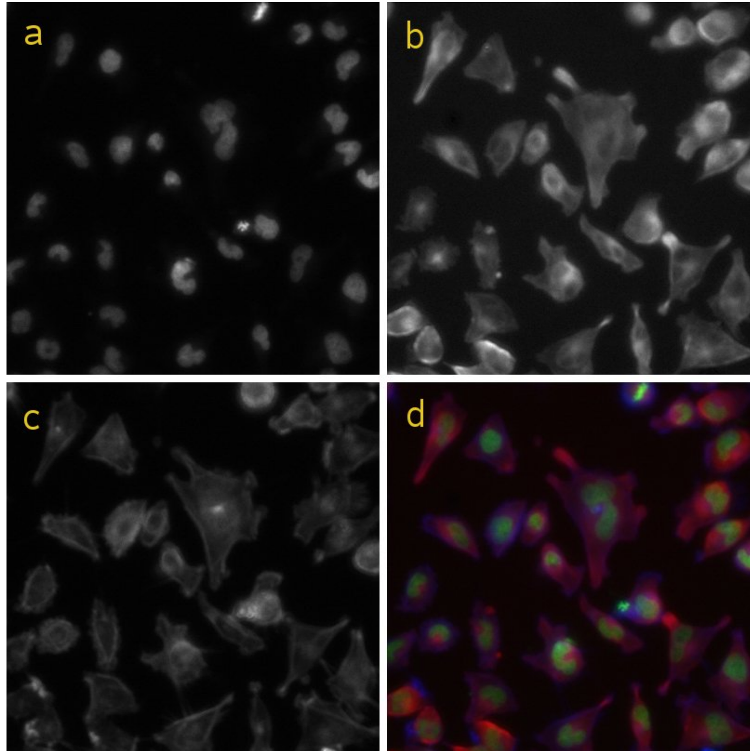

Figure 2: Composing a false-colour image (d) from a set of grayscale microscopy images for three different luminescent compounds: (a) – DAPI, (b) – tubulin and (c) – phalloidin

The following code demonstrates how grayscale images recorded using three different microscope filters (Figure 2 **a**, **b** and **c**) can be put together into a single *false-colour* representation (Figure 2 **d**), and conversely, how a single false-colour image can be decomposed into its individual channels.

```
> files <- c("ch1.png","ch2.png","ch3.png")
> orig <- read.image(files, Grayscale)
> abc <- orig[150:550, 120:520,]
> a <- channel(abc[,,1], "asgreen")  # RGB
> b <- channel(abc[,,2], "asred")    # RGB
> d <- a + b + channel(abc[,,1], "asblue")
> C <- channel(d, "blue")            # gray
```

Analysing an RNAi screen

Consider an experiment in which images like in Figure 2 were recorded for each of $\approx$ 20,000 genes, using a whole-genome RNAi library to test the effect of gene knock-down on cell viability and appearance. Among the image descriptors of interest are the number, position, size, shape and the fluorescent intensities of cells and nuclei.

As a side effect of the watershed transform, the function `watershed`, EBImage identifies separable

objects in an image. For this particular purpose it is advisable to use the result of the distance map transform (`distmap`) as input. Object descriptors can be obtained by calling `getObjects` on the result. One can see that an image has data of detected object by printing it (typing its name and hitting `Enter`): this information will be included in the object description. The features are a list of matrices, one per image frame, each containing different objects in its rows and different object descriptors in the columns. If a reference grayscale image was supplied to the watershed routine, object gray value intensities will be calculated as well.

For every gene, the image analysis workflow looks, therefore, as follows: load and normalize images, perform image segmentation, enhance the segmented images by morphological opening and closing, generate distance maps and use them to identify cells and nuclei and to extract image descriptors, and, finally, generate image previews with the identified objects marked.

Object descriptors can then be analysed statistically to cluster genes by their phenotypic effect, generate a list of genes that should be studied further in more detail (hit list), e. g. genes that have a specific phenotypic effect of interest, etc. The image previews can be used to verify and audit the performance of the algorithm through visual inspection.

A schematic implementation is illustrated in the following example code and in Figure 3. Here we omit the step of nuclei detection (object `x1`), from where the matrix of nuclei coordinates (object `seeds`) is retrieved to serve as starting points for the cell detection. The nuclei detection is done analogously to the cell detection without specifying starting points, seeds.

```
> for (X in genes) {
+  files <- dir(pattern=X)
+  orig <- read.image(files)
+  abc <- normalize(orig)
+  i1 <- abc[,,1]
+  i2 <- abc[,,2]
+  i3 <- abc[,,3]
+  a <- sqrt(normalize(i1 + i3))
+  b <- thresh( blur(a, 4, 2), 50, 50)
+  C <- opening(b, morphKern(7))
+  C <- closing(C, morphKern(7))
+  d <- distmap(C)  # shown normalized
+ ## x1 <- watershed(...) # nuclei detection
+  features(x1) <- getObjects(x1, i1)
+  x2 <- propagate(x1, a, d)
+  features(x2) <- getObjects(x2, i3)
+  rgb <- channel(i1, "asgreen") + channel(i2, "asred")
+  rgb <- rgb + channel(i3, "asblue")
+  e <- paintObjects(x2, rgb)
+  f <- paintObjects(x1, paintObjects(x2, i3) )
+ }
```

Note that here we adopted the *record-at-a-time* approach: image data, which can be huge, are stored on a mass-storage device and are loaded into RAM in portions of just a few images at a time.

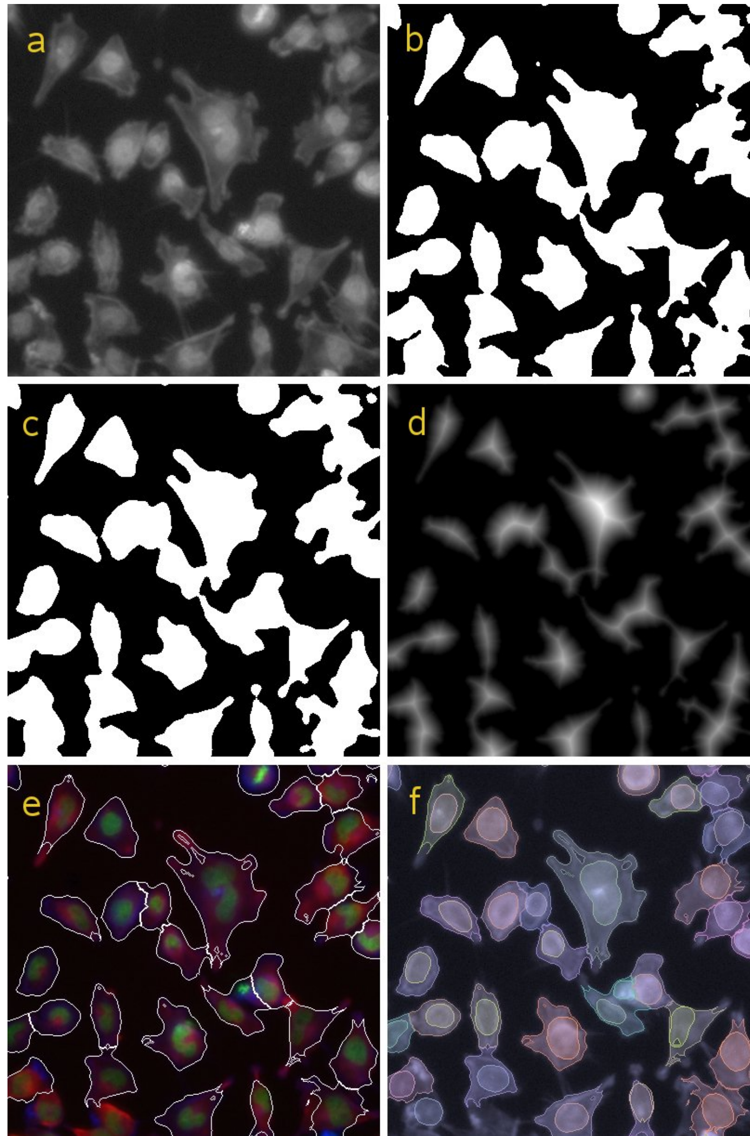

Figure 3: Illustration of the object detection algorithm: (a) – *sqrt*-brightened combined image of nuclei (DAPI from Figure 2**a**) and cells (phalloidin from Figure 2**c**); (b) – image **a** after *blur* and *adaptive thresholding*; (c) – image **b** after morphological *opening* followed by *closing*; (d) – normalized *distance map* generated from image **c**; (e) – outlines of cells detected using `watershed` drawn on top of the RGB image from Figure 2**d**; (f) – colour-mapped cells and nuclei as detected with `watershed` (one unique colour per object)

Summary

EBImage brings image processing and image analysis capabilities to R. Its focus is the programmatic (non-interactive) analysis of large sets of similar images, such as those that are obtained in cell-based assays for gene function via RNAi knock-down. Image descriptors extracted as the result of analysis can be analysed further using existing R-functionality in machine learning (clustering, classification) and hypothesis testing.

Our future developments in image analysis will focus primarily on more accurate object detection and on algorithms for feature/descriptor extraction, and eventually on image registration, alignment and object tracking. Algorithms for the statistical analysis of image descriptors will be developed as part of a separate package that uses EBImage for image processing and analysis. In addition, one can imagine many other useful features, for example, support for more ImageMagick functions. Contributions or collaborations on these or other topics are welcome.

Acknowledgements and Availability

We thank F. Fuchs and M. Boutros for providing their miscroscopy data and for many stimulating discussions about the technology and the biology, and the European Bioinformatics Institute (EBI), Cambridge, UK, for financial support.

EBImage is available from BioConductor (`www.bioconductor.org`). The package was tested to work on 32 and 64 bit UNIX and Linux systems, on both PowerPC and Intel based MacOS and on MS Windows. The package requires external dependencies, ImageMagick and GTK+, therefore, please read the included file 'INSTALL' in the package tarball for installation instructions on different systems.

Parts of this text for older versions of EBImage were used in R-News [Sklyar2006].

References

[Sklyar2006] O. Sklyar, W. Huber. Image Analysis for microscopy Screens. *R-News*, 6/5:12–16, 2006.

[Boutros2004] M. Boutros, A. Kiger, S. Armknecht, *et al.* Genome-wide RNAi analysis of cell growth and viability in Drosophila. *Science*, 303:832–835, 2004.

[Carpenter2004] A. E. Carpenter and D.M. Sabatini. Systematic genome-wide screens of gene function. *Nature Reviews Genetics*, 5:11–22, 2004.

[ImageMagick] ImageMagick: software to convert, edit, and compose images. *Copyright: ImageMagick Studio LLC*, 1999-2006. URL `http://www.imagemagick.org/`

[Moffat2006] J. Moffat and D.M. Sabatini. Building mammalian signalling pathways with RNAi screens. *Nature Reviews Mol. Cell Biol.*, 7:177–187, 2006.

[Neumann2006] B. Neumann, M. Held, U. Liebel, *et al.* High-throughput RNAi screening by time-lapse imaging of live human cells. *Nature Mathods*, 3(5):385–390, 2006.

[Russ2002] J. C. Russ. The image processing handbook – 4th ed. CRC Press, *Boca Raton*. 732 p., 2002

[Wiemann2004] S. Wiemann, D. Arlt, W. Huber, *et al.* From ORFeome to biology: a functional genomics pipeline. *Genome Res.* 14(10B):2136–2144, 2004.