



# Introduction to **EBImage**, an image processing and analysis toolkit for R

Gregoire Pau, Oleg Sklyar, Wolfgang Huber  
[gpau@ebi.ac.uk](mailto:gpau@ebi.ac.uk)

September 3, 2009

## Contents

<b>1</b>	<b>Reading/displaying/writing images</b>	<b>1</b>
<b>2</b>	<b>Image objects and matrices</b>	<b>3</b>
<b>3</b>	<b>Spatial transformations</b>	<b>4</b>
<b>4</b>	<b>Color management</b>	<b>5</b>
<b>5</b>	<b>Image filtering</b>	<b>6</b>
<b>6</b>	<b>Drawing on images</b>	<b>7</b>
<b>7</b>	<b>Morphological operations</b>	<b>8</b>
<b>8</b>	<b>Segmentation</b>	<b>9</b>
<b>9</b>	<b>Object manipulation</b>	<b>11</b>
<b>10</b>	<b>Cell segmentation example</b>	<b>12</b>

## 1 Reading/displaying/writing images

The package `EBImage` is loaded by the following command.

```
> library("EBImage")
```

The function `readImage` is able to read images from files or URLs. The supported image formats depend on the capabilities of the underlying `ImageMagick` library.

```
> f = system.file("images", "lena.gif", package="EBImage")  
> lena = readImage(f)
```

Images can be displayed using the function `display`. Pixel intensities should range from 0 (black) to 1 (white).



Figure 1: lena, lenac

```
> display(lena)
```

Color images or images with multiple frames can also be read with `readImage`.

```
> lenac = readImage(system.file("images", "lena-color.png", package="EBImage"))
> display(lenac)
> nuc = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
> display(nuc)
```

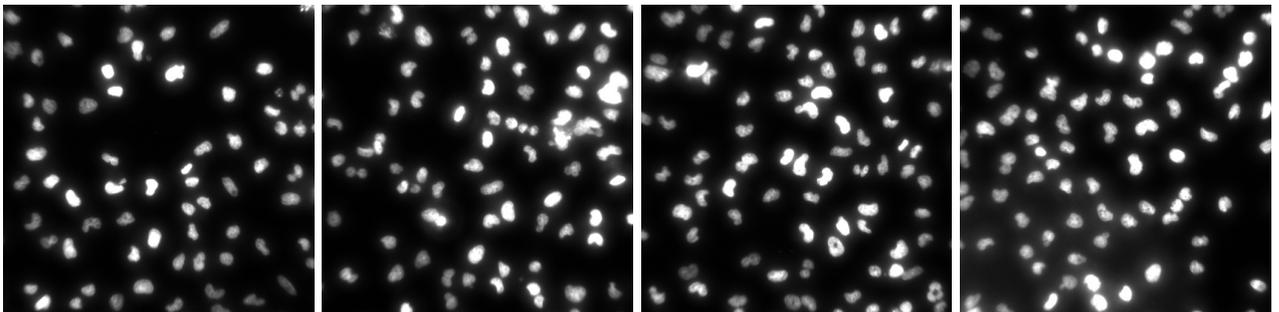


Figure 2: nuc

Images can be written with `writeImage`. The file format is deduced from the file name extension. This is useful to convert image formats, here from PNG format to JPEG format.

```
> writeImage(lena, 'lena.jpeg', quality=95)
> writeImage(lenac, 'lenac.jpeg', quality=95)
```

## 2 Image objects and matrices

The package `EImage` uses the class `Image` to store and process images. Images are stored as multi-dimensional arrays containing the pixel intensities. All `EImage` functions are also able to work with matrices and arrays.

```
> print(lena)
```

```
Image
```

```
  colormode: Grayscale
  storage.mode: double
  dim: 512 512
  nb.total.frames: 1
  nb.render.frames: 1
```

```
imageData(object)[1:5,1:6]:
```

```
      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.5372549 0.5372549 0.5372549 0.5372549 0.5372549 0.5490196
[2,] 0.5372549 0.5372549 0.5372549 0.5372549 0.5372549 0.5490196
[3,] 0.5372549 0.5372549 0.5372549 0.5372549 0.5372549 0.5137255
[4,] 0.5333333 0.5333333 0.5333333 0.5333333 0.5333333 0.5098039
[5,] 0.5411765 0.5411765 0.5411765 0.5411765 0.5411765 0.5333333
```

As matrices, images can be manipulated with all R mathematical operators. This includes `+` to control the brightness of an image, `*` to control the contrast of an image or `^` to control the gamma correction parameter.

```
> lena1 = lena+0.5
```

```
> lena2 = 3*lena
```

```
> lena3 = (0.2+lena)^3
```



Figure 3: `lena`, `lena1`, `lena2`, `lena3`

Others operators include `[]` to crop images, `<` to threshold images or `t` to transpose images.

```
> lena4 = lena[299:376, 224:301]
```

```
> lena5 = lena>0.5
```

```
> lena6 = t(lena)
```

```
> print(median(lena))
```

```
[1] 0.3803922
```

Images with multiple frames are created using `combine` which merges images.

```
> lenacomb = combine(lena, lena*2, lena*3, lena*4)
```

```
> display(lenacomb)
```



Figure 4: lena, lena4, lena5, lena6



Figure 5: lenacomb

### 3 Spatial transformations

Specific spatial image transformations are done with the functions `resize`, `rotate`, `translate` and the functions `flip` and `flop` to reflect images.

```
> lena7 = rotate(lena, 30)
> lena8 = translate(lena, c(40, 70))
> lena9 = flip(lena)
```



Figure 6: lena, lena7, lena8, lena9

## 4 Color management

The class `Image` extends the base class `array` and uses the `colormode` slot to store how the color information of the multi-dimensional data should be handled.

As an example, the color image `lenac` is a 512x512x3 array, with a `colormode` slot equals to `Color`. The object is understood as a color image by `EImage` functions.

```
> print(lenac)
```

```
Image
```

```
  colormode: Color
  storage.mode: double
  dim: 512 512 3
  nb.total.frames: 3
  nb.render.frames: 1
```

```
imageData(object)[1:5,1:6,1]:
```

```
      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.8862745 0.8862745 0.8862745 0.8862745 0.8862745 0.8901961
[2,] 0.8862745 0.8862745 0.8862745 0.8862745 0.8862745 0.8901961
[3,] 0.8745098 0.8745098 0.8745098 0.8745098 0.8745098 0.8901961
[4,] 0.8745098 0.8745098 0.8745098 0.8745098 0.8745098 0.8705882
[5,] 0.8862745 0.8862745 0.8862745 0.8862745 0.8862745 0.8862745
```

The function `colorMode` can access and change the value of the slot `colormode`, modifying the rendering mode of an image. In the next example, the `Color` image `lenac` with one frame is changed into a `Grayscale` image with 3 frames, corresponding to the red, green and blue channels. The function `colorMode` does not change the content of the image but changes only the way the image is rendered by `EImage`.

```
> colorMode(lenac) = Grayscale
> display(lenac)
```



Figure 7: `lenac`, rendered as a `Color` image and as a `Grayscale` image with 3 frames (red channel, green channel, blue channel)

The color mode of image `lenac` is reverted back to `Color`.

```
> colorMode(lenac) = Color
```

The function `channel` performs colorspace conversion and can convert `Grayscale` images into `Color` ones both ways and can extract color channels from `Color` images. Unlike `colorMode`, `channel` changes the pixel intensity values of the image. The function `rgbImage` is able to combine 3 `Grayscale` images into a `Color` one.

```
> lenak = channel(lena, 'rgb')
> lenak[236:276, 106:146, 1] = 1
> lenak[236:276, 156:196, 2] = 1
> lenak[236:276, 206:246, 3] = 1
> lenab = rgbImage(red=lena, green=flop(lena), blue=flop(lena))
```



Figure 8: lenak, lenab

## 5 Image filtering

Images can be linearly filtered using `filter2`. `filter2` convolves the image with a matrix filter. Linear filtering is useful to perform low-pass filtering (to blur images, remove noise, ...) and high-pass filtering (to detect edges, sharpen images, ...). Various filter shapes can be generated using `makeBrush`.

```
> flo = makeBrush(21, shape='disc', step=FALSE)^2
> flo = flo/sum(flo)
> lenaflo = filter2(lenac, flo)
> fhi = matrix(1, nc=3, nr=3)
> fhi[2,2] = -8
> lenafhi = filter2(lenac, fhi)
```

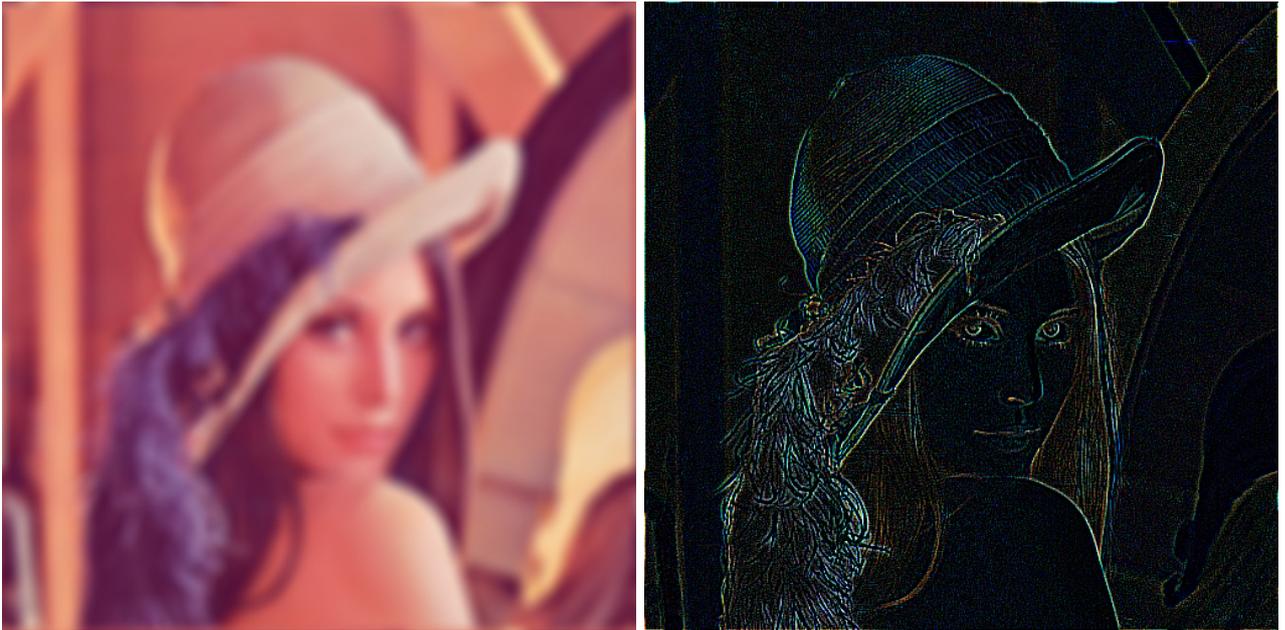


Figure 9: Low-pass filtered `lenafl0` and high-pass filtered `lenafhi`

## 6 Drawing on images

The function `drawtext` draws text labels on images.

```
> font = drawfont(weight=600, size=28)
> lenatext = drawtext(lena, xy=c(250, 450), labels="Lena", font=font, col="white")
```



Figure 10: `lenatext`

## 7 Morphological operations

Binary images are images where the pixels of value 0 constitute the background and the other ones constitute the foreground. These images are subject to several non-linear mathematical operators called morphological operators, able to erode and dilate an image.

```
> ei = readImage(system.file('images', 'shapes.png', package='EBImage'))
> ei = ei[110:512,1:130]
> display(ei)
> kern = makeBrush(5, shape='diamond')
> eierode = erode(ei, kern)
> eidilat = dilate(ei, kern)
```



Figure 11: `ei` ; `eierode` ; `eidilat`

## 8 Segmentation

Segmentation consists in extracting objects from an image. The function `bwlabel` is a simple function able to extract every connected sets of pixels from an image and relabel these sets with a unique increasing integer. `bwlabel` can be used on binary images and is useful after thresholding.

```
> eilabel = bwlabel(ei)
> cat('Number of objects=', max(eilabel), '\n')
```

Number of objects= 7

```
> nuct = nuc[:,1]>0.2
> nuclabel = bwlabel(nuct)
> cat('Number of nuclei=', max(nuclabel), '\n')
```

Number of nuclei= 74



Figure 12: `ei`, `eilabel/max(eilabel)`

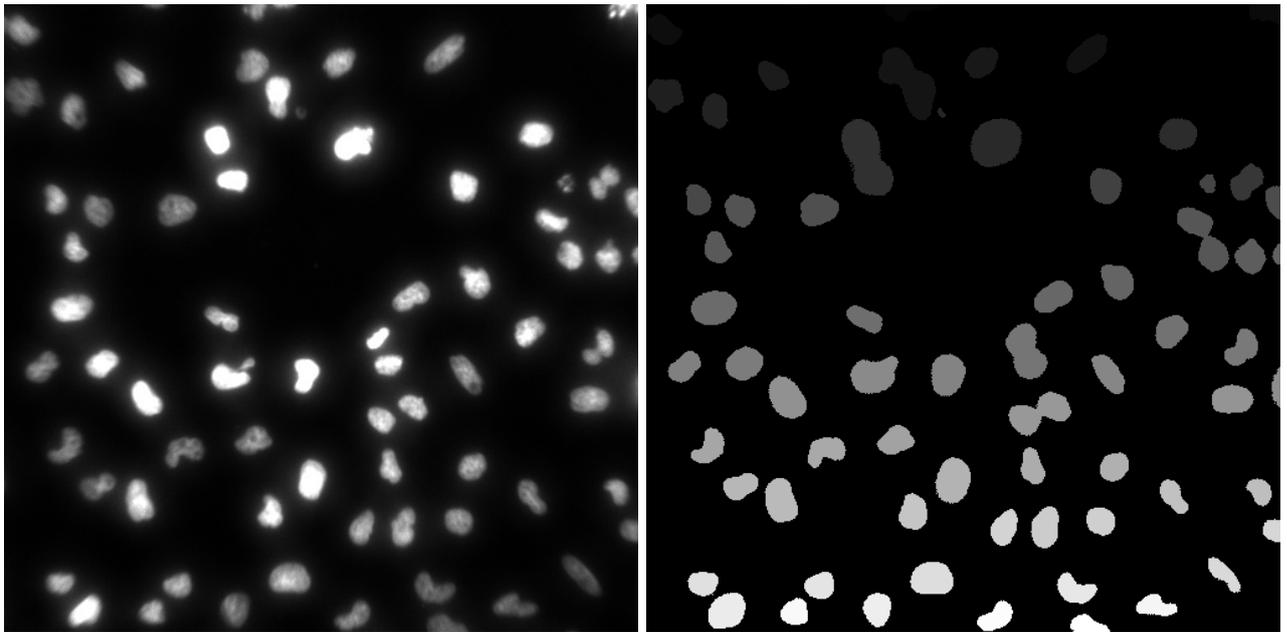


Figure 13: `nuc[ , ,1]`, `nuclabel/max(nuclabel)`

Since the images `eilabel` and `nuclabel` range from 0 to the number of object they contain (given by `max(eilabel)` and `max(nucabel)`), they have to be divided by these number before displaying, in order to fit the `[0,1]` range needed by `display`.

The grayscale top-bottom gradient observable in `eilabel` and `nuclabel` is due to the way `bwlabel` labels the connected sets, from top-left to bottom-right.

Adaptive thresholding consists in comparing the intensity of pixels with their neighbors, where the neighborhood is specified by a filter matrix. The function `thresh` performs a fast adaptive thresholding of an image with

a rectangular window while the combination of `filter2` and `<` allows a finer control. Adaptive thresholding allows a better segmentation when objects are close together.

```
> nuct2 = thresh(nuc[:,1], 10, 10, 0.05)
> kern = makeBrush(5, shape='disc')
> nuct2 = dilate(erode(nuct2, kern), kern)
> nuclabel2 = bwlabel(nuct2)
> cat('Number of nuclei=', max(nuclabel2), '\n')
```

Number of nuclei= 76

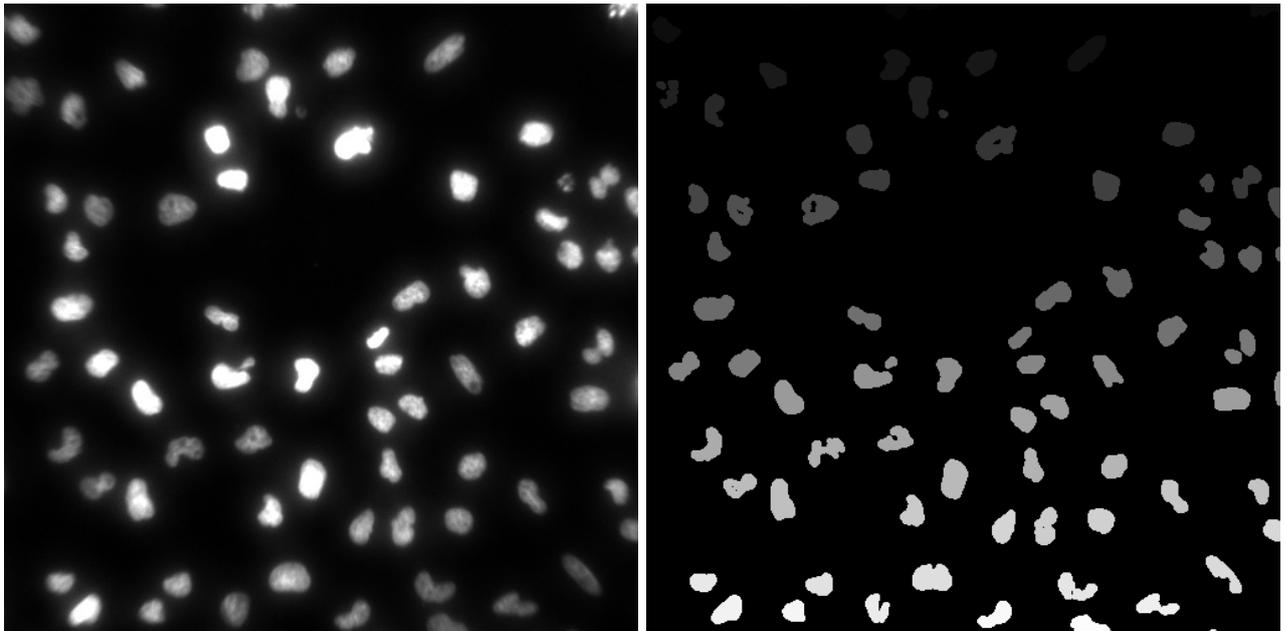


Figure 14: `nuc[ , ,1]`, `nuclabel2/max(nuclabel)`

## 9 Object manipulation

Objects, defined as sets of pixels with the same unique integer value can be outlined and painted using `paintObjects`. Some holes are present in objects of `nuclabel12` which can be filled using `fillHull`.

```
> nucgray = channel(nuc[, , 1], 'rgb')
> nuch1 = paintObjects(nuclabel12, nucgray, col='#ff00ff')
> nuclabel13 = fillHull(nuclabel12)
> nuch2 = paintObjects(nuclabel13, nucgray, col='#ff00ff')
```

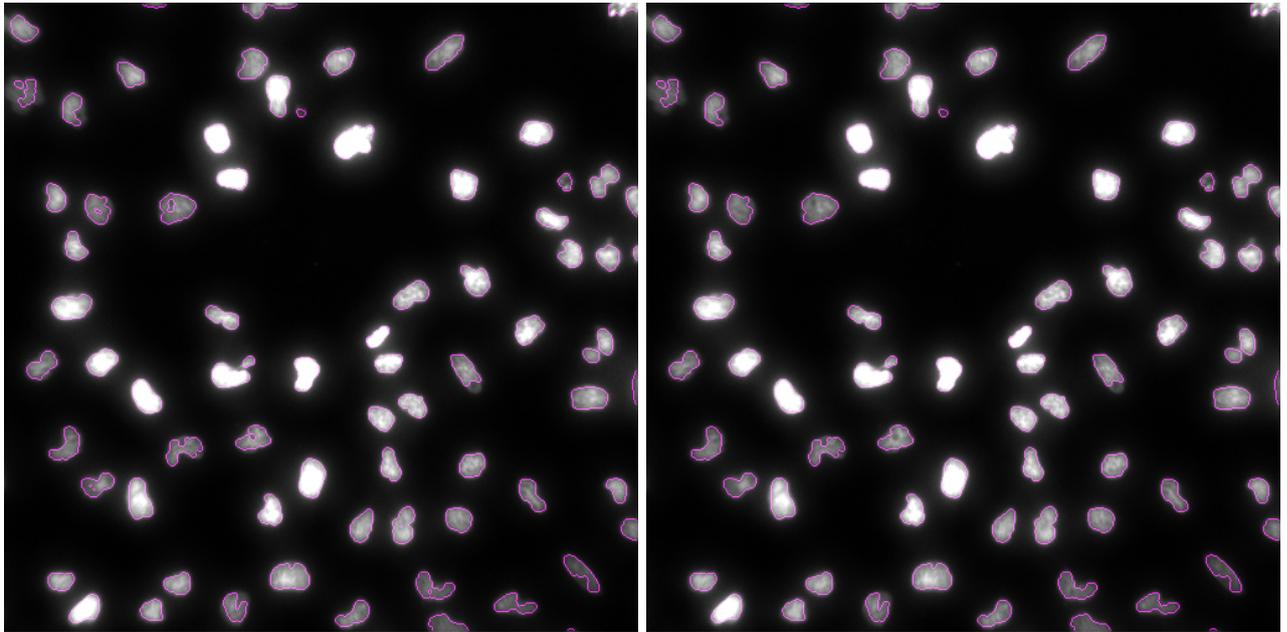


Figure 15: `nuch1`, `nuch2`

Objects features like hull features, shape descriptors, moments, centers, texture descriptors are extracted using `hullFeatures`, `moments` and `getFeatures` for more complex features. Nucleus centers are here extracted to obtain nuclei coordinates, which are used to draw the nucleus indexes.

```
> xy = hullFeatures(nuclabel13)[, c('g.x', 'g.y')]
> font = drawfont(weight=600, size=16)
> nuctext = drawtext(nuch2, xy=xy, labels=as.character(1:nrow(xy)), font=font, col="yellow")
```

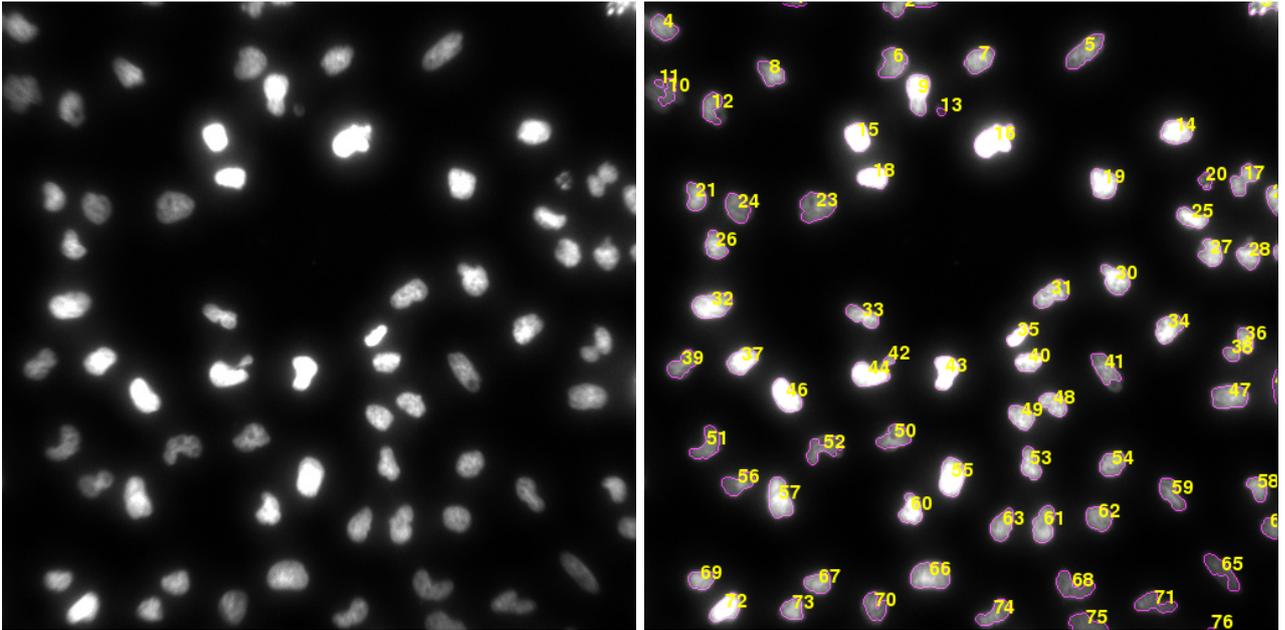


Figure 16: Original nuc, segmented nuctext

## 10 Cell segmentation example

This is a complete example of segmentation of cells (nucleus + cell bodies) using the functions described before and the function `propagate`, able to perform Voronoi-based region segmentation.

Images of nuclei and cell bodies are first loaded:

```
> nuc = readImage(system.file('images', 'nuclei.tif', package='EBImage'))
> cel = readImage(system.file('images', 'cells.tif', package='EBImage'))
> img = rgbImage(green=1.5*cel, blue=nuc)
```

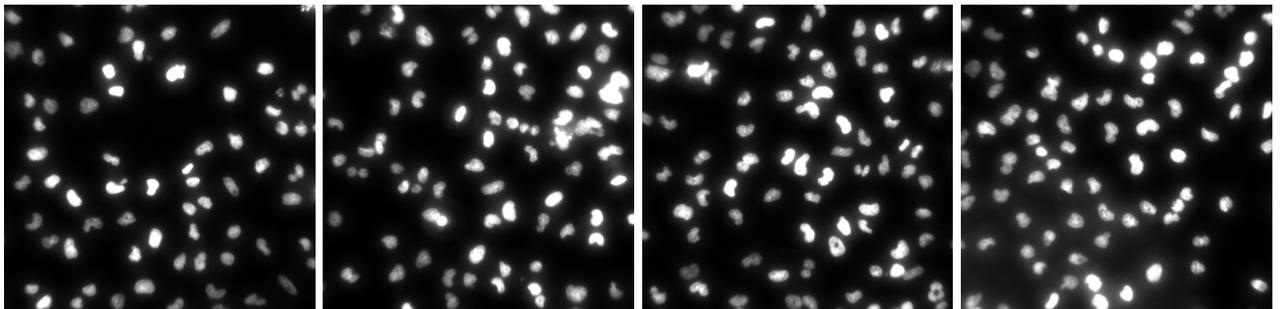


Figure 17: nuc

Nuclei are first segmented using `thresh`, `fillHull`, `bwlabel` and `opening`, which is an erosion followed by a dilatation.

```
> nmask = thresh(nuc, 10, 10, 0.05)
> nmask = opening(nmask, makeBrush(5, shape='disc'))
> nmask = fillHull(nmask)
> nmask = bwlabel(nmask)
```

Cell bodies are segmented using `propagate`.

```
> ctmask = opening(cel > 0.1, makeBrush(5, shape='disc'))
> cmask = propagate(cel, nmask, ctmask)
```

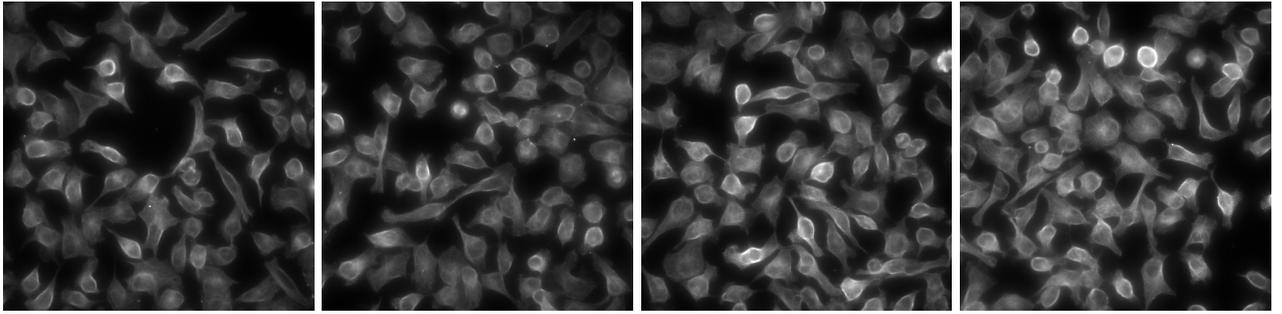


Figure 18: cel1

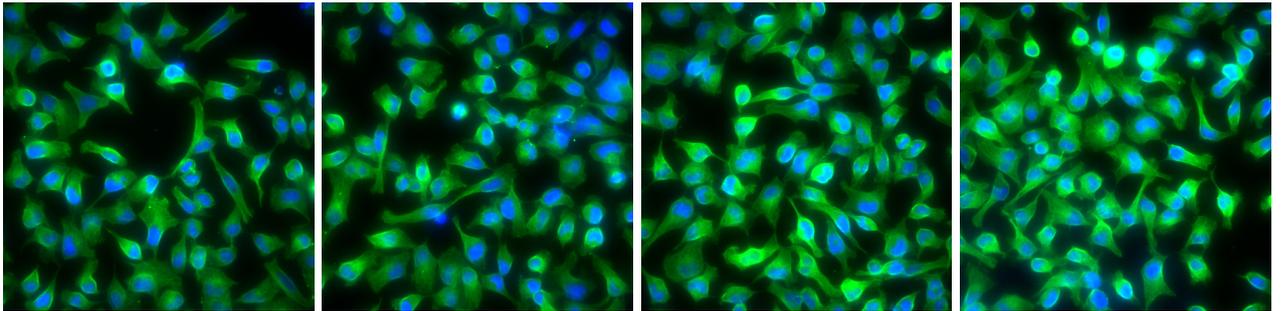


Figure 19: img

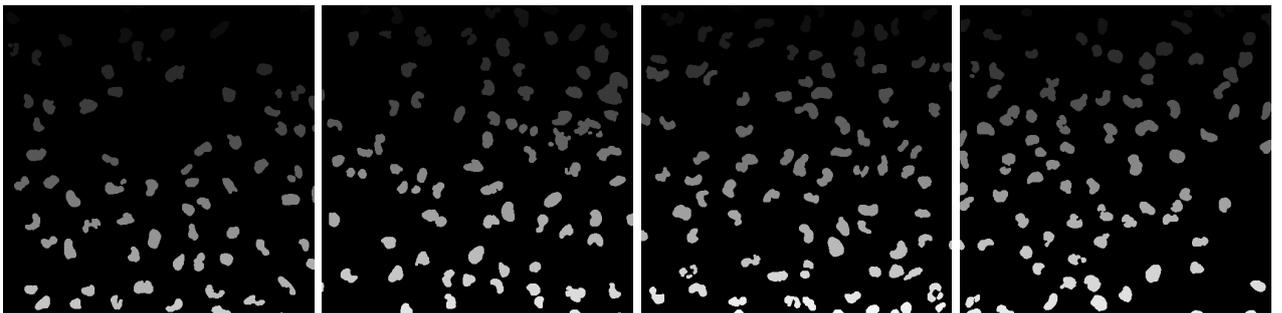


Figure 20: nmask/max(nmask)

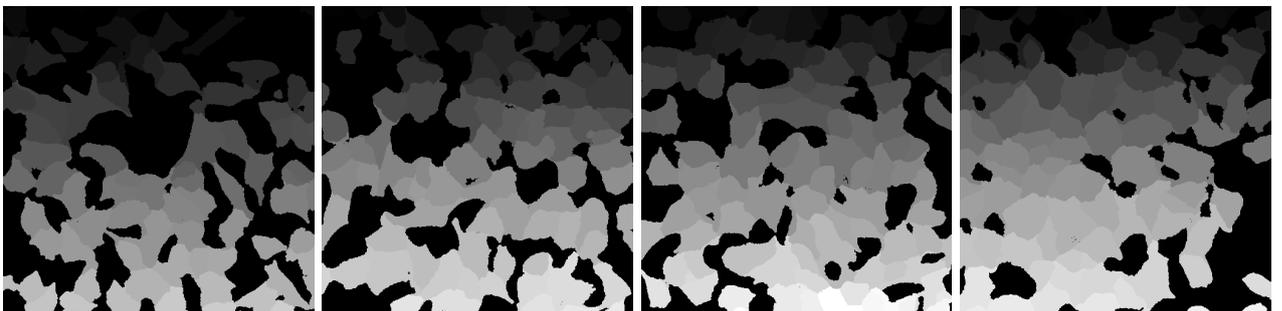


Figure 21: cmask/max(cmask)

Cells are outlined using `paintObjects` and labels are printed using `hullFeatures` and `drawtext`.

```
> res = paintObjects(cmask, img, col='#ff00ff')
> res = paintObjects(nmask, res, col='#ffff00')
```

```

> xy = lapply(hullFeatures(cmask), function(hf) hf[, c('g.x', 'g.y')])
> labels = lapply(xy, function(z) as.character(1:nrow(z)))
> font = drawfont(weight=600, size=16)
> res = drawtext(res, xy=xy, labels=labels, font=font, col="white")

```

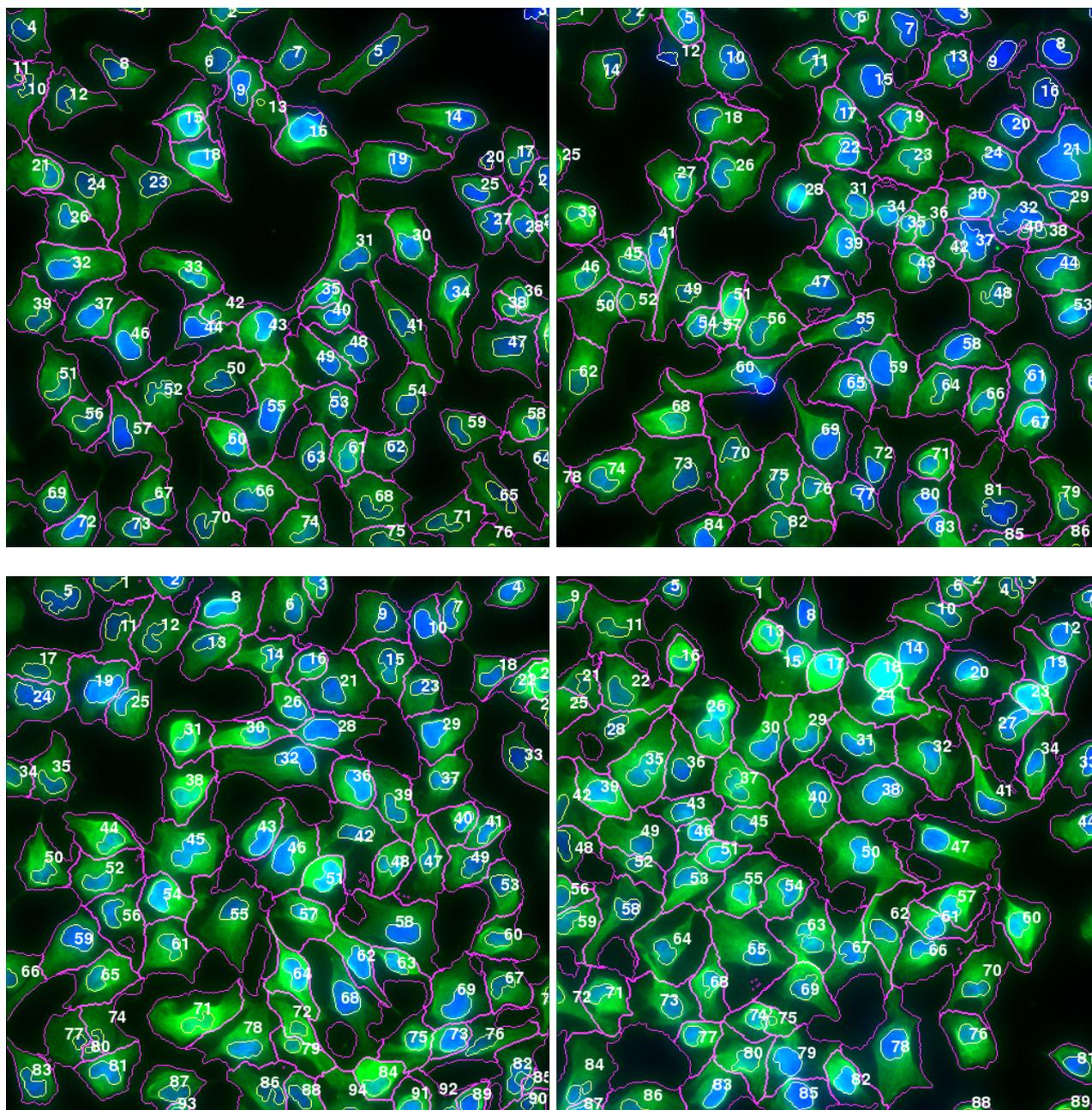


Figure 22: Final segmentation res