CRImage
a package for classifying cells and calculating tumour cellularity

Henrik Failmezger*, Yinyin Yuan, Oscar Rueda, Florian Markowetz

∗ E-mail: failmezger@mpipz.mpg.de

Contents

1 Load the package

The package is loaded by the following command:

```
> library(CRImage)
```

See the functions of the package EBImage to read, write and manipulate an image.

2 Image processing

2.1 Color space conversion

Images can be converted to the HSV and LAB color space.

```
> f = system.file("extdata", "exImg2.jpg", package="CRImage")
> img=readImage(f)
> #convert image to the HSV color space
```

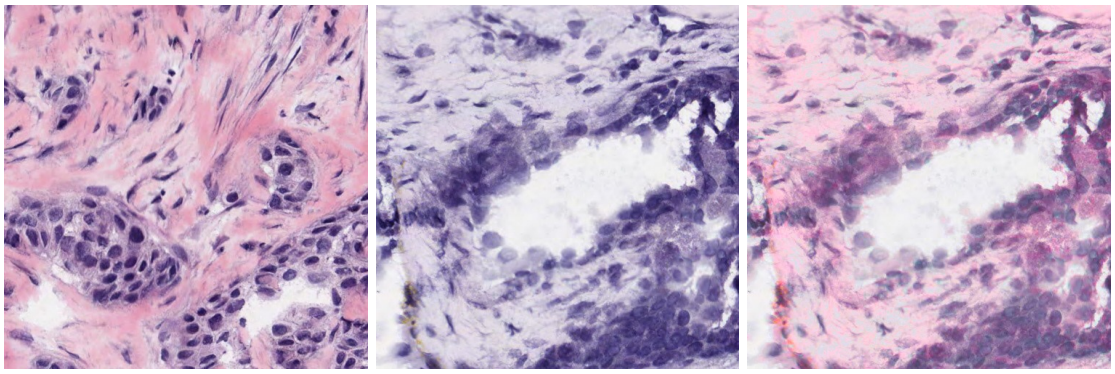

Figure 1. Color correction. The color values of the image in the middle are adapted to the color values of the left image.

```
> imgHSV=convertRGBToHSV(img)
> #convert image to the LAB color space
> imgLAB=convertRGBToLAB(img)
> #convert back to the RGB color space
> imgRGB=convertHSVToRGB(imgHSV)
> imgRGB=Image(imgRGB)
> colorMode(imgRGB)='color'
> #display(imgRGB)
>
> #convert back to the RGB color space
> imgRGB=convertLABToRGB(imgLAB)
> imgRGB=Image(imgRGB)
> colorMode(imgRGB)='color'
> #display(imgRGB)
```

2.2 Color correction

In order to correct for staining deviations the color space of one image can be adapted to the color space of a target image. The target image is converted to the LAB color space and the mean and standard deviation of every channel is calculated. Afterwards the image is standardized to the mean and standard deviation of the target image.

3 Image thresholding

The function `createBinaryImage` can be used to create a binary image of a grayscale image. Either Otsu thresholding or Phansalkar thresholding can be used for this task. Otsu thresholding trys to separate the grayscale histogram in two parts. Phansalkar thresholding calculates the threshold based on the mean and standard deviation of the values in a certain window. The threshold is calculated on a RGB grayscale image and an Euclidean distance image of the LAB colour space, the results are ORed afterwards.

This function calculates Otsu threshold of a grayscale image.

```
> f = system.file("extdata", "exImg2.jpg", package="CRImage")
> img=readImage(f)
> #convert to grayscale
```

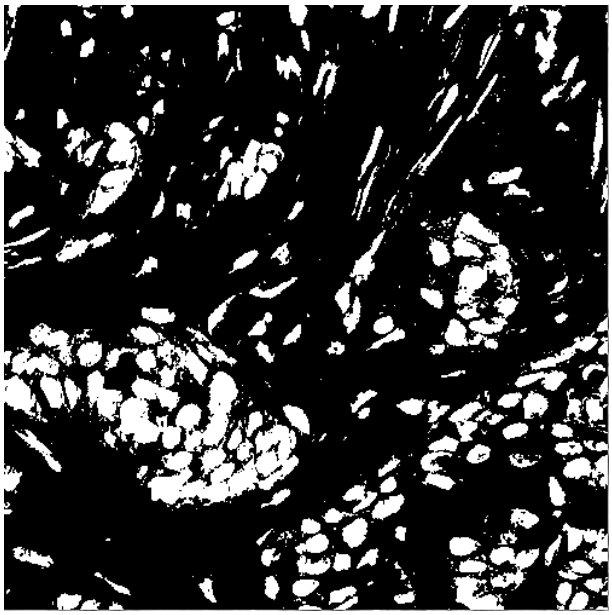

Figure 2. Thresholded image. Cell nuclei are coloured white, background is coloured black.

```
> imgG=EBImage::channel(img,"gray")
> #create a mask for white pixel
> whitePixelMask=img[,,1]>0.85 & img[,,2]>0.85 & img[,,3]>0.85
> #create binary image
> imgB=createBinaryImage(imgG,img,method="otsu",numWindows=4,whitePixelMask=whitePixelMask)
```

The image is read with `readImage`. `whitePixelMask` is TRUE for white pixels these pixels are considered as background and excluded from thresholding. The image is converted to grayscale by the function `channel(exImgB,"gray")` of EBImage. The function `createBinaryImage` creates a binary image using the thresholding method "otsu".

4 Segment an image

An image can be segmented to find cells in the image, using the function `segmentImage`.

```
> f = system.file("extdata", "exImg2.jpg", package="CRImage")
> segmentationValues=segmentImage(filename=f,maxShape=800,minShape=40,failureRegion=2000,threshold="c
```

The image is converted to grayscale and thresholded. Morphological opening is used to delete clutter and to smooth the shapes of the cells. The watershed algorithm is used to separate clustered cells. The parameter `maxShape` defines the maximal shape of cell nuclei. Segmented nuclei which exceed this value will be thresholded and segmented again. The parameter `minShape` defines the minimum size if cell nuclei. Cell nuclei, which fall below this value will be deleted. The parameter `failureRegion` defines, when artifacts in the image should be deleted. Dark regions which exceed this value will be deleted. The list element `segmentationValues[[1]]` holds the original image, `segmentationValues[[2]]` holds the segmented image and `segmentationValues[[3]]` holds features, which were calculated for the segmented objects. The segmented objects can be drawn by the function `display` (see EBImage).

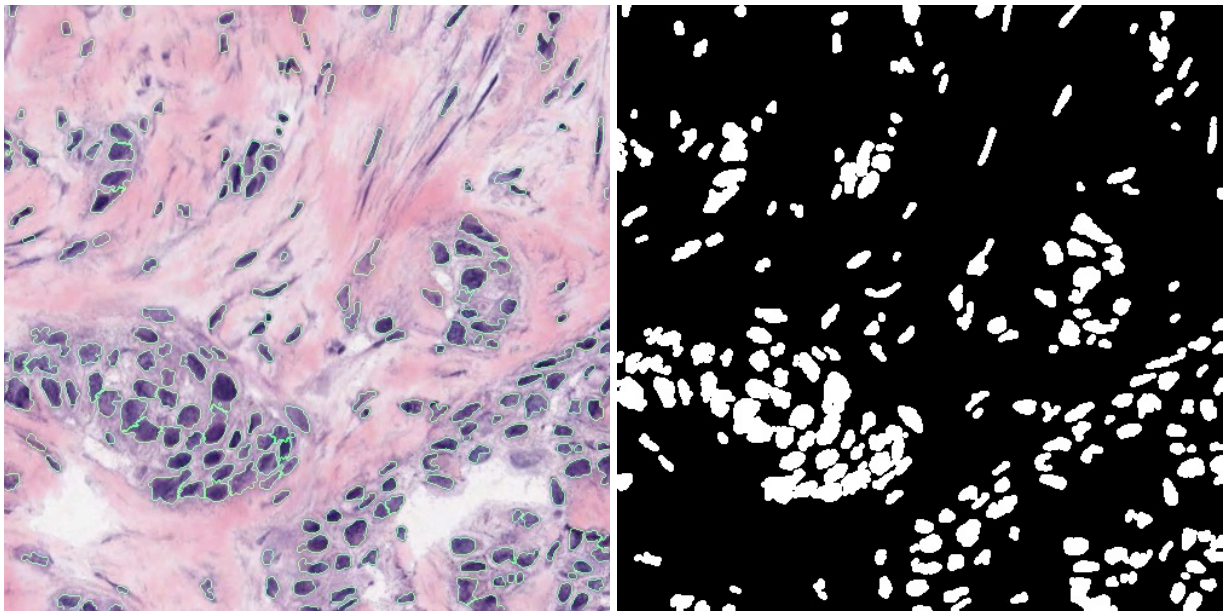

Figure 3. SegmentedImage. Cell nuclei are labeled in green.

5 Creating a training set

To classify cells in images first an appropriate training set has to be created.

```
f = system.file("extdata", "exImg.jpg", package="CRImage")
trainingValues=createTrainingSet(filename=f,maxShape=800,minShape=40,failureRegion=2000)
```

The list trainingValues returns two values. The first value is an image in which every segmented cell is numbered. The second value is a table with features for every cell:

```
index class      g.x       g.y g.s g.p     g.pdm    g.pdsd    g.effr...
1   <NA>148.2203  4.855932 118  36  5.721049 1.0130550  6.128668...
2   <NA>  160.2719  4.763158 114  38  5.659780 1.0331399  6.023896...
3   <NA>  183.7975  3.101266  79  35  4.593585 0.9575905  5.014627...
4   <NA>  196.3500  4.242857 140  43  6.424591 1.4433233  6.675581...
5   <NA>  271.5694  2.680556  72  29  4.504704 1.2490794  4.787307...
6   <NA>  338.5221  6.530973 113  35  5.601531 1.0849651  5.997418...
7   <NA>  456.0179  2.946429 112  39  5.726556 1.8101368  5.970821...
8   <NA>  556.3778  9.018519 270  81  9.196894 2.4687870  9.270581...
9   <NA>  575.1777 10.935950 484 101 11.959995 2.1378649 12.412171...
10   <NA>  592.7391  1.724638  69  41  5.145094 2.2782539  4.686511...
```

To create the training set, class values for the cells have to be inserted in the column "class":

```
index class      g.x       g.y g.s g.p     g.pdm    g.pdsd    g.effr
1   normal 148.2203  4.855932 118  36  5.721049 1.0130550  6.128668
2   malignant  160.2719  4.763158 114  38  5.659780 1.0331399  6.023896
```

The values in the column "index" match the numbers for the cells in the image. You can save the table as tab seperated by:

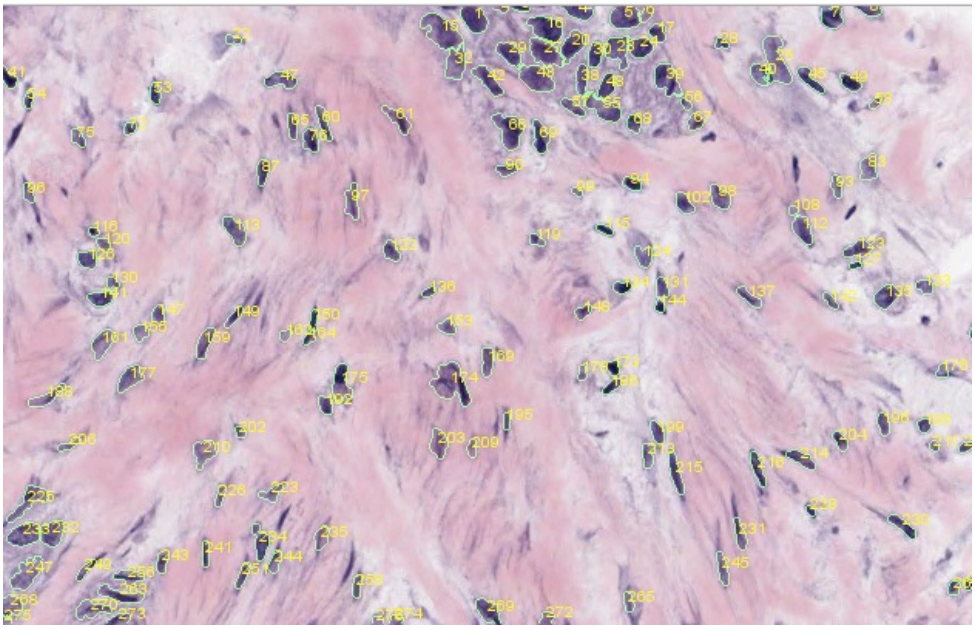

Figure 4. The image with labeled cell nuclei. The number of the cell nuclei correspond to the index in the feature table.

```
write.table(trainingData,file="path",sep="\t",rownames=F)
```

and open it for example in a spreadsheet program, however be careful not to shift the columns. You do not have to assign a class value to every cell, but do ensure that there are enough examples for every class. Class values can be numbers (e.g. 1,2,3) or strings (e.g. "normal", "malignant"). You can choose at most 10 classes. If you want to use the kernel smoothing approach for classification you can only choose two classes or you have to specifiy a cancer class, when classifying cells.

6 Creating the classifier

The command:

```
> f = system.file("extdata", "trainingData.txt", package="CRImage")
> #read training data
> trainingData=read.table(f,header=TRUE)
> #create classifier
> classifierValues=createClassifier(trainingData)
> classifier=classifierValues[[1]]
> #classifiedCells=classifierValues[[2]]
> #display(classifiedCells)
```

creates the classifier. The classifier is a Support Vector Machine provided by the package e1017.

7 Classification of cells

After having created the classifier, cells in another image can be classified (Figure 3).

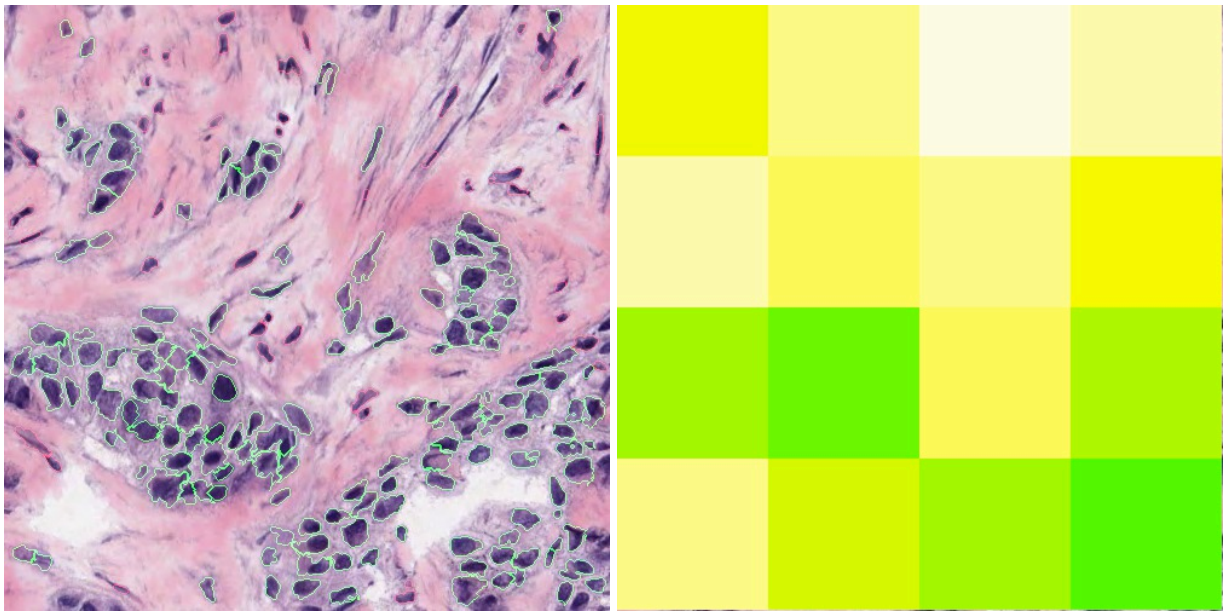

Figure 5. Classified Image (left) and heatmap of cellularity values (right). Malignant cells are colored green, whereas other cells are coloured red. The heatmap of tumour cellularity indicates regions of large tumour cellularity (strong green values) and low tumour cellularity (white regions).

```
>         #classify cells
>         f = system.file("extdata", "exImg2.jpg", package="CRImage")
>         classValues=classifyCells(classifier,filename=f,KS=TRUE,maxShape=800,minShape=40,failureReg
```

8 Calculation of cellularity

If tumour images are processed, the cellularity of the tumour can be calculated. The image is first segmented and the cell types are classified. Afterwards, the cellularity of the tumour is determined. The function needs to know, which class value should be the tumour class.

```
> t = system.file("extdata", "trainingData.txt", package="CRImage")
> #read training data
> trainingData=read.table(t,header=TRUE)
> #create classifier
> classifier=createClassifier(trainingData)[[1]]
> #calculation of cellularity
> f = system.file("extdata", "exImg2.jpg", package="CRImage")
> cellularity=calculateCellularity(classifier=classifier,filename=f,KS=TRUE,maxShape=800,minShape=40,
```

9 Classification of Aperio Image Slides

Large pathological images are often difficult to process due to their often large file size. Images of ScanScope TX scanner can be saved in the CWS file format. In this format the images are separated in smaller subimages. These images can be process with the function `processAperio`.

```
> #create the classifier
> t = system.file("extdata", "trainingData.txt", package="CRImage")
> trainingData=read.table(t,header=TRUE)
> classifier=createClassifier(trainingData)[[1]]

> dir.create("AperiOutput")
> f = system.file("extdata",  package="CRImage")
> processAperio(classifier=classifier,inputFolder=f,outputFolder="AperiOutput",identifier="Da",numSec
```

The function gets a classifier, which was for instance created with `createClassifier`. An input folder has to be specified, which includes the subimages. The common identifier of the subimages has tp specified (either "Da" or "Ss", depending on the adjustments of the Aperio software). If the image includes different sections for which cellularity has to be calculated separately, this can be specified with `numSections`, the function will afterwards do an automatical clustering of the subimages to their corresponding sections. The function needs to know, which class of the classifier is the tumour class, which is specified with `cancerIdentifier`. The function creates three folders in the output folder. The folder classifiedImage includes the classified subimages. The folder `Files` includes files with cellularity values for every section. The folder `tumourDensity` includes cancer heatmaps for every subimage.

10 Copy Number Correction:

Segmentation of copy number and correction of log-ratios (LRR) and beta allele frequency (BAF) values for cellularity.

```
> LRR <- c(rnorm(100, 0, 1), rnorm(10, -2, 1), rnorm(20, 3, 1),
+        rnorm(100,0, 1))
> BAF <- c(rnorm(100, 0.5, 0.1), rnorm(5, 0.2, 0.01), rnorm(5, 0.8, 0.01), rnorm(10, 0.25, 0.1), rnor
+        rnorm(100,0.5, 0.1))
> Pos <- sample(x=1:500, size=230, replace=TRUE)
> Pos <- cumsum(Pos)
> Chrom <- rep(1, length(LRR))
> z <- data.frame(Name=1:length(LRR), Chrom=Chrom, Pos=Pos, LRR=LRR, BAF=BAF)
> res <- correctCopyNumber(arr="Sample1", chr=1, p=0.75, z=z)
```

The results of `correctCopyNumber` can be plotted using `plotCorrectedCN`:

```
>         plotCorrectedCN(res, chr=1)
```