Analysis of high-throughput microscopy-based screens with imageHTS

Gregoire Pau, Xian Zhang, Michael Boutros, and Wolfgang Huber

gregoire.pau@embl.de

January 13, 2022

Contents

1 Introduction

imageHTS is an R package dedicated to the analysis of high-throughput microscopy-based screens. The package provides a modular and extensible framework to segment cells, extract quantitative cell features, predict cell types and browse screen data through web interfaces. Designed to operate in distributed environments, imageHTS provides a standardized access to remote screen data, facilitating the dissemination of high-throughput microscopy-based screens.

In the following, we first show how to use imageHTS to analyse a microscopy-based RNA interference (RNAi) screen by automated cell segmentation and extraction of morphological cell features. In a second example, we demonstrate how to access and analyse data from a remote screen repository.

2 Analysis of a microscopy-based screen

The `kimorph` screen is an RNAi screen where HeLa cells were fixed 48 h after siRNA transfection and stained for DNA, tubulin and actin. The screen assays 800 siRNAs and is described in [1]. In this section, we are analyzing a 12-well subset of this screen, of reduced image quality (due to package size considerations), located in the `inst/submorph` directory of the imageHTS package.

2.1 Initialization

In imageHTS, screen data files can be accessed in two locations: in a local repository, indicated by `localPath`, or in an optional remote server designated by `serverURL`. If a file is not present in the local repository, e.g. for storage capacity reasons, imageHTS automatically retrieves the corresponding file from the remote server to the local repository. This dual repository feature is useful when screen data is stored in a different location from where it is analysed.

After loading the package imageHTS, we initialize an imageHTS object with `parseImageConf`. The function takes 3 arguments: an imageHTS configuration file and the variables `localPath` and `serverURL`. The imageHTS configuration file, in DCF format, describes the general screen configuration: where the microscopy images are located and how the plates and wells are named. We are using the imageHTS configuration file shown in section 2.6. A detailed description of the imageHTS configuration file can be found in the manual pages of `parseImageConf`. We set the variable `localPath` to a temporary directory, for storing intermediate analysis files. The variable `serverURL` can point either to a directory or an external URL. In the following example, `serverURL` points to the submorph screen data directory of the imageHTS package, which contains the source images acquired from the microscope.

```
> library('imageHTS')
```

```
> localPath = tempdir()
> serverURL = system.file('submorph', package='imageHTS')
> x = parseImageConf('conf/imageconf.txt', localPath=localPath,
+                    serverURL=serverURL)

File "conf/imageconf.txt" read.
Number of plates= 1
Number of replicates= 2
```

```
Number of wells= 384
Number of channels= 3
Number of spots= 1
```

The imageHTS object `x` is now instantiated. The function `configure` configures the screen by providing the screen description, the plate configuration layout (how sample, control and empty wells are located in the plates) and the screen log. The function `annotate` sets up the mapping between reagents and gene targets. Both functions originate from the package cellHTS2, dedicated to the analysis of low-content RNAi screens [2]. The imageHTS class extends the cellHTS class and both functions are fully compatible with their cellHTS2 counterparts. See cellHTS2 documentation for details.

```
> x = configure(x, 'conf/description.txt', 'conf/plateconf.txt',
+                  'conf/screenlog.txt')
> x = annotate(x, 'conf/annotation.txt')
```

In imageHTS, each well is uniquely referred by an unique ID. Well unique IDs are generated by the function `getUnames`, which can filter wells according to their plate, replicate, row, column or content type (as described in the plate configuration file). The following example enumerates the wells that are not empty.

```
> unames = setdiff(getUnames(x), getUnames(x, content='empty'))
> unames

 [1] "001-01-A03" "001-01-A05" "001-01-B03" "001-01-B05" "001-01-C03"
 [6] "001-01-D03" "001-02-A03" "001-02-A05" "001-02-B03" "001-02-B05"
[11] "001-02-C03" "001-02-D03"
```

12 wells are non-empty in this screen. Metadata (plate, replicate, content, gene target, annotation) about the wells is retrieved using the function `getWellFeatures`.

```
> getWellFeatures(x, unames[1:3])

           plate well controlStatus PlateName Content     siRNAID GeneID
001-01-A03     1  A03           ubc        P1 control        <NA>    UBC
001-01-A05     1  A05        sample        P1  sample M-005300-00   AAK1
001-01-B03     1  B03           ubc        P1 control        <NA>    UBC
           LocusID Accession
001-01-A03    <NA>      <NA>
001-01-A05   22848 NM_014911
001-01-B03    <NA>      <NA>
```

2.2 Cell segmentation

Cells present in wells can be segmented using the function `segmentWells`. `segmentWells` is a high-level function that takes a vector of unique well IDs and a DCF segmentation parameter file. `segmentWells` uses the low-level segmentation function indicated by the field `seg.method` of the segmentation parameter file to segment individual well images. For each well, `segmentWells` writes in the local directory: calibrated image data 'cal', segmentation data 'seg' and several JPEG images. Files can be accessed later on with the functions `fileHTS` and `readHTS`, as shown in the sequel.

If an unique well is given, `segmentWells` returns a list of three images: a calibrated image, a nucleus mask and a cell mask. The images can be manipulated with the package EBImage [3] and visualized using the command `display`. The function `highlightSegmentation` merges the calibrated image, the nucleus and cell masks to produce a composite image that highlights the segmentation information.

In the following, we segment the third negative control well `rluc` using the segmentation parameter file shown in section 2.6. The field `seg.method` of the file indicates the function `segmentATH` to segment the well. This function is specifically designed to segment cells stained for DNA and cytoskeletal proteins but any other segmentation function can be used, e.g. for segmenting yeast cells in bright field images or segmenting organelles stained with specific markers.

```
> uname = getUnames(x, content='rluc')[3]
> print(uname)

[1] "001-02-C03"

> y =  segmentWells(x, uname=uname,
+                       segmentationPar='conf/segmentationpar.txt')

001-02-C03: rccussfs nbcells=88 OK

> display(y$cal)
> hseg = highlightSegmentation(0.6*y$cal, y$nseg, y$cseg, thick=TRUE)
> display(hseg)
```

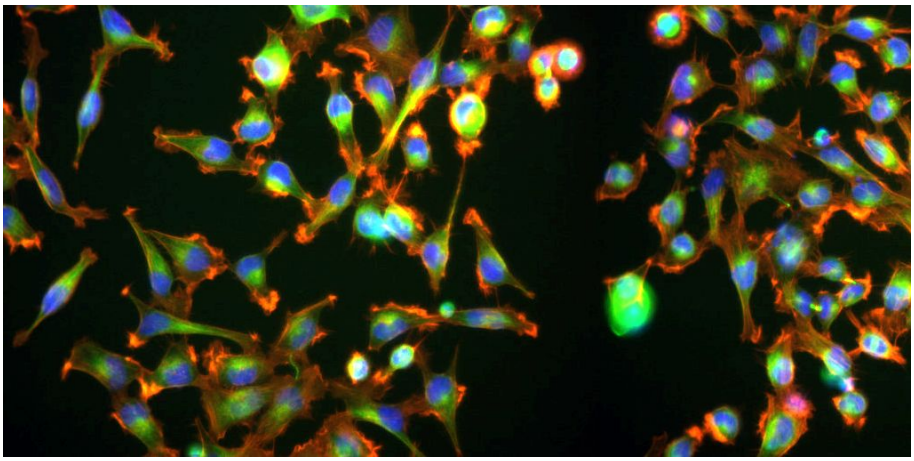

Figure 1: Calibrated image 'y$cal' from well '001-02-C03'

Segmentation of the full screen is done with the following commands and takes about 4 minutes with a single processor. Since wells can be segmented independently from each other, segmentation of the full screen can be easily parallelized using many processors. The following example is not run in this vignette, due to time constraints.

```
> unames = setdiff(getUnames(x), getUnames(x, content='empty'))
> segmentWells(x, unames, 'conf/segmentationpar.txt')
```

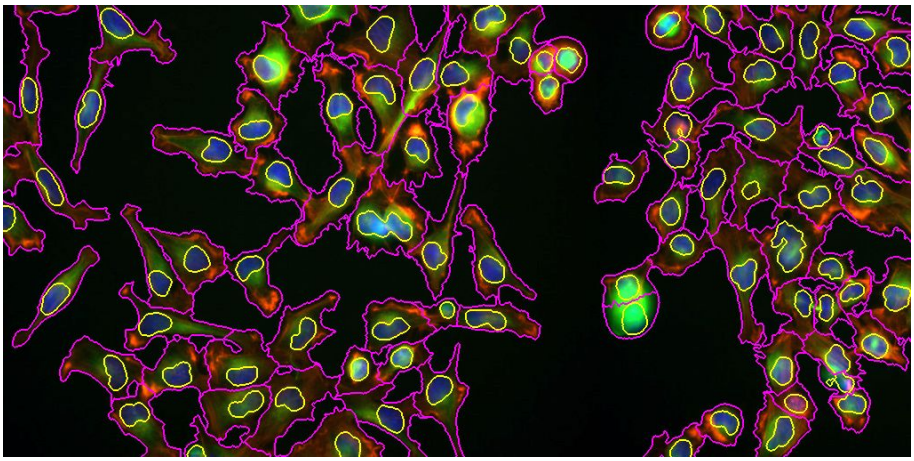

Figure 2: Segmented image 'hseg' from well '001-02-C03'
Cell nucleus is highlighted in yellow and cell membrane is indicated in magenta.

In imageHTS, all screen data files can be accessed through the function `fileHTS`, including configuration files, source images, segmentation data, cell features and JPEG images. `fileHTS` creates paths pointing to screen data files, using a standardized naming scheme. The following example shows, for the well indicated by `uname`, how to get access to first channel of the source image, calibrated image data, and the JPEG image of the well.

```
> fileHTS(x, type='source', uname=uname, channel=1)

[1] "/tmp/RtmpreblpB/source/PK-11B-pl1/Well-C003/Tritc.jpeg"

> fileHTS(x, type='seg', uname=uname)

[1] "/tmp/RtmpreblpB/data/001-02/001-02-C03_seg.rda"

> fileHTS(x, type='viewfull', uname=uname)

[1] "/tmp/RtmpreblpB/view/001-02/001-02-C03_full.jpeg"
```

2.3 Quantification of cell features

Quantification of cell features is done by the high-level function `extractFeatures` on a set of wells, using a feature parameter file. Similar to the function `segmentWells`, `extractFeatures` uses the function indicated by the field `extractfeatures.method` of the feature parameter file to extract cell features. For each well, `extractFeatures` writes features in the local directory, in tab-separated format. In the following example, we extract cell features from the well indicated by `uname`, using the feature parameter file shown in section 2.6.

```
> extractFeatures(x, uname, 'conf/featurepar.txt')

001-02-C03: gmbhc OK
```

Cell features can be accessed using the function `fileHTS`, as described above. However, for convenience purposes, the function `readHTS` combines `fileHTS` and reads the corresponding file, according to the specified format (here, tab-separated). The following example reads the cell feature matrix of well '001-02-C03'.

```
> y = readHTS(x, type='ftrs', uname=uname, format='tab')
> dim(y)

[1]  88 293

> y[1:10, 1:7]

   spot id c.s.area c.s.perimeter c.s.radius.mean c.s.radius.sd c.s.radius.min
1     1  1     3214           317        33.82418      8.647554       13.78002
2     1  2     2205           183        26.59331      5.793902       11.99122
3     1  3     1693           171        24.15556      5.847740       12.15843
4     1  4     4560           323        39.69377      7.712418       26.55506
5     1  5     3363           296        33.96740      8.939805       19.10263
6     1  6     3513           246        35.24268      8.472181       19.57477
7     1  7     5111           310        40.48104      7.407180       26.45049
8     1  8     4199           312        37.14185      5.575376       27.07191
9     1  9     3931           330        38.06178     12.982466       18.37205
10    1 10     5815           365        43.39953      5.527656       31.82344
```

88 cells are present in the well and each cell is described with 293 features. Cell features include geometrical features, moment-based features, Haralick moments and Zernicke features. Cell features are described in the manual pages of the function `getFeatures` of the package EBImage. Some features have a direct interpretation, such as `c.s.area`, which measures the cell area or `c.t.b.mean`, which quantifies the cell tubulin mean intensity. In the following example, we display the distribution of the latter within the cells of the well, and identify the cells that have a tubulin intensity higher than 1600.

```
> ctub <- y$c.t.b.mean*y$c.s.area
> hist(ctub, 20, xlab='Cell tubulin intensity (a.u.)', main='')
> abline(v=1600, col=2)
> cellid = which(ctub>1600)
> print(cellid)

[1]  2  8 10 15 72
```

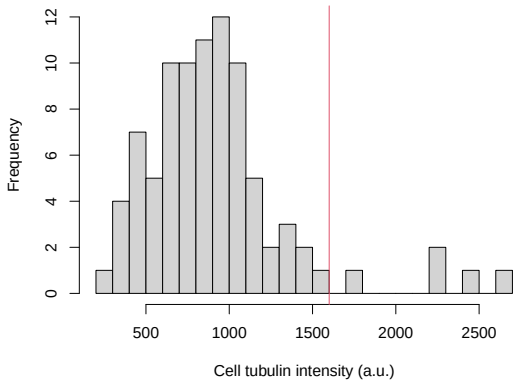

Figure 3: Distribution of cell tubulin intensity in cells of well '001-02-C03'

Five cells have a tubulin content higher than 1600. Since rows of cell feature matrix are synchronised with cell indexes in segmentation masks, cells can be easily traced back by loading the segmentation information, as shown in the following example.

```
> cal = readHTS(x, type='cal', uname=uname, format='rda')
> seg = readHTS(x, type='seg', uname=uname, format='rda')
> cseg = rmObjects(seg$cseg, setdiff(1:nrow(y), cellid))
> hightub = highlightSegmentation(0.6*cal, cseg=cseg, thick=TRUE)
> display(hightub)
```

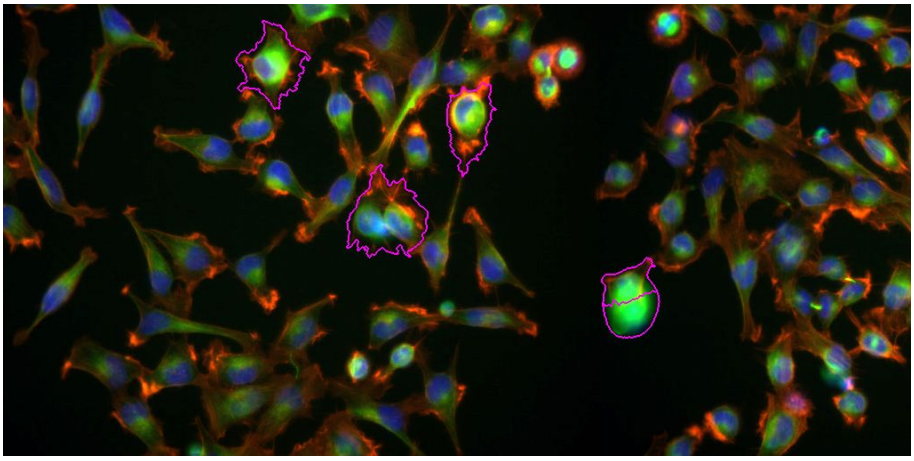

Figure 4: Cells of well '001-02-C03' having a tubulin intensity higher than 1600

2.4 Prediction of cell classes

Cell features can be used as covariates to classify cells, using supervised learning and a set of manually annotated cells. The function `readLearnTS` takes as arguments a training set file and the feature parameter file, previously used in `extractFeatures`. The training set is a list of labelled cells and the feature parameter file contains the field `remove.classification.features`, indicating the features that should not be used during training/classification (e.g. cell position). Construction of the training set is done using the annotation web module cellPicker as described in the section 3.2.

The function `readLearnTS` uses a Support Vector Machine with a radial kernel to predict cell labels. Training is done by parameter grid-search and 5-fold cross-validation, to minimize classification error. The function creates the file `data/classifier.rda`, which contains the trained classifier. The following example trains a cell classifier, but is not run in the vignette due to time constraints.

```
> set.seed(1)
> readLearnTS(x, 'conf/featurepar.txt', 'conf/trainingset.txt')
```

After training, prediction of cell labels is done by the function `predictCellLabels`. The function writes for each well a vector of predicted cell labels. The following example predicts the cell labels of the well '01-02-C03', using a classifier previously trained on a set of 66 cells labelled with 3 cell classes: I (interphase), M (mitotic) and D (debris).

```
> predictCellLabels(x, uname)

001-02-C03: D=18 I=67 M=3 OK
```

67 interphase, 3 mitotic and 18 debris cells were predicted in the image. The following example retrieves and displays the predicted cell labels.

```
> clab = readHTS(x, type='clabels', uname=uname, format='tab')
> labid = split(1:nrow(clab), clab$label)
> inter = seg$cseg%in%labid$I
> mito = seg$cseg%in%labid$M
> debris = seg$cseg%in%labid$D
> dc = Image(c(inter+mito, inter, debris+inter), colormode='Color',
+            dim=c(dim(seg$cseg)[1:2], 3))
> dc = highlightSegmentation(0.5*dc+0.2*drop(cal), cseg=seg$cseg,
+                            thick=TRUE)
> display(dc)
```

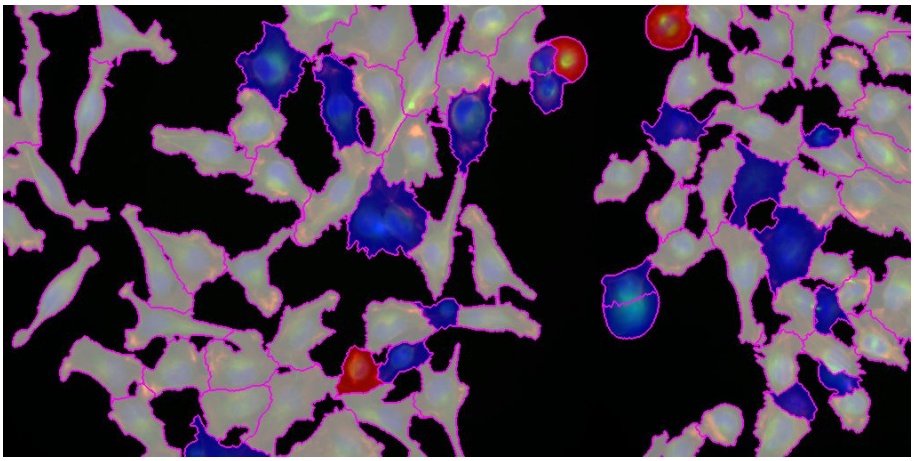

Figure 5: Predicted cell labels (grey: interphase, red: mitotic, blue: debris) in well '001-02-C03'

Overall prediction is very good, except for few cells. Classification performance can be easily improved by enlarging the training set and re-run the training and predicting steps. The cellPicker web module, described in section 3.2, has an interactive cell annotation interface which is very useful to refine the training set.

2.5 Phenotype summarization

Cell population features are summarized by `summarizeWells`. The function computes for each well a phenotypic profile, which summarizes cell population features. Currently, a phenotypic profile consist of: cell number `n`, median cell feature `med.*` (for each feature) and cell class ratios. `summarizeWells` creates the file `data/profiles.tab` which contains the phenotypic profiles. The following example computes the phenotypic profiles of all the wells, but is not run in the vignette due to time constraints.

```
> summarizeWells(x, unames, 'conf/featurepar.txt')
```

In the following example, the phenotypic profiles (previously computed and stored in the imageHTS package) are loaded with `readHTS` and averaged by well type. Only the following features are considered: n (cell number), med.c.s.area (median cell size), med.c.t.b.mean (median cell tubulin density), M (mitotic cell fraction) and D (debris cell fraction).

```
> profiles = readHTS(x, type='file', filename='data/profiles.tab',
+                    format='tab')
> wfcontent =
+   factor(as.character(getWellFeatures(x, unames)$controlStatus))
> table(wfcontent)

wfcontent
  rluc sample    ubc
     4      4      4

> zwf = split(1:nrow(profiles), wfcontent)
> ft = c('n', 'med.c.s.area', 'med.c.t.b.mean', 'M', 'D')
> avef = do.call(rbind,
+                lapply(zwf, function(z) colMeans(profiles[z, ft])))
> print(avef)

           n med.c.s.area med.c.t.b.mean           M         D
rluc   99.00     2446.750      0.2496445 0.010388399 0.1573825
sample 86.75     2793.375      0.2324918 0.031074095 0.1455122
ubc    26.75     1576.250      0.4220933 0.005813953 0.7868731
```

There are 4 `rluc` negative controls, 4 `ubc` positive controls and 4 `sample` wells in this screen. The average number of cells in `ubc` wells is 26.75, lower than in `rluc` wells, 99.00. Moreover, the average fraction of debris cells in `ubc` wells, 0.79, is higher than in `rluc` wells, 0.16. A larger number of replicates and proper statistical testing would be needed to determine whether the observed changes are statistically significant.

2.6 Configuration files and complete script

Configurations files used in this vignette are reproduced in this section. Since the files are part of the screen data, they can be read using `fileHTS`. In the following example, we display the imageHTS configuration file, the segmentation parameter file and the feature parameter file.

```
> f = fileHTS(x, 'file', filename='conf/imageconf.txt')
> cat(paste(readLines(f), collapse='\n'), '\n')

AssayName: submorph
SourceFilenamePattern: source/PK-{replicate}B-pl{plate}/Well-{row}0{col}/{channel}.jpeg
PlateNames: 1
ReplicateNames: 10, 11
RowNames: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P
ColNames: 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
ChannelNames: Tritc, FITC, Hoechst
```

```
> f = fileHTS(x, 'file', filename='conf/segmentationpar.txt')
> cat(paste(readLines(f), collapse='\n'), '\n')
```

```
seg.method: segmentATH

nuc.athresh.filter: makeBrush(35, shape='box')/(35*35)
nuc.athresh.t: 0.00424
nuc.morpho.kernel: makeBrush(3, shape='diamond')
nuc.watershed.tolerance: 3
nuc.watershed.neighbourood: 2
nuc.min.density: 0.1
nuc.min.size: 125.0625
nuc.max.size: 2070.25

adj.a: 2.82*a - 0.17
adj.t: 5.03*t - 0.35
adj.h: 2.99*h - 0.15

cell.thresh.filter: matrix(c(0,1,0,1,2,1,0,1,0)/6,nc=3,nr=3)
cell.thresh.t: 0.12
cell.morpho.kernel: makeBrush(3, shape='diamond')
cell.propagate.lambda: 0.0001
cell.propagate.mix.power: 0.2
cell.min.density: 0.1
cell.max.edgepratio: 0.3
cell.min.size: 150.0625
cell.max.size: 14491.75
cell.max.perimeter: 769.3

thumbnail.crop: 100, 600, 200, 400
thumbnail.resize.width: 200
```

```
> f = fileHTS(x, 'file', filename='conf/featurepar.txt')
> cat(paste(readLines(f), collapse='\n'), '\n')

extractfeatures.method: getCellFtrsATH
cell.classes: D, I, M
remove.classification.features: c.a.m.cx, c.a.m.cy, c.t.m.cx, c.t.m.cy, c.h.m.cx, c.h.m.cy, c.m.m.cx, c.m.m.c
cellHTS.features: n, med.c.s.area, med.c.m.m.eccentricity, med.n.ah.cor, M
cellHTS.features.name: Number of cells, Median cell size, Median cell ecc., Median A/H nuc. corr., Metaphase
```

The following example is the complete script used to automatically segment cells, quantify cell features, predict cell labels and summarize phenotypes of the whole screen. The example is not run in this vignette, due to time constraints.

```
> library('imageHTS')
> localPath = tempdir()
> serverURL = system.file('submorph', package='imageHTS')
> x = parseImageConf('conf/imageconf.txt', localPath=localPath,
+                    serverURL=serverURL)
> x = configure(x, 'conf/description.txt', 'conf/plateconf.txt',
+                  'conf/screenlog.txt')
> x = annotate(x, 'conf/annotation.txt')
> unames = setdiff(getUnames(x), getUnames(x, content='empty'))
> segmentWells(x, unames, 'conf/segmentationpar.txt')
```

```
> extractFeatures(x, unames, 'conf/featurepar.txt')
> readLearnTS(x, 'conf/featurepar.txt', 'conf/trainingset.txt')
> predictCellLabels(x, unames)
> summarizeWells(x, unames, 'conf/featurepar.txt')
```

3 Getting access to remote screen data

The dual repository architecture of imageHTS allows an easy access to remote screen data. In the following, we are analysing the full `kimorph` RNAi screen, targeting about 800 protein coding genes in HeLa cells. Screen details are available in [1]. The screen has been previously analysed by imageHTS and screen data is located at http://www.huber.embl.de/cellmorph/kimorph/. The interactive webQuery browsing interface is available at http://www.huber.embl.de/cellmorph/kimorph/webquery/.

3.1 Initialization

We first initialize an imageHTS object by setting the variable `serverURL` to the screen data URL and the local repository `localPath` to an empty local directory. We next configure and annotate the imageHTS objects using the screen configuration files. The files, absent in the local screen directory, are automatically downloaded from the remote server.

```
> localPath = file.path(tempdir(), 'kimorph')
> serverURL = 'http://www.huber.embl.de/cellmorph/kimorph/'
> x = parseImageConf('conf/imageconf.txt', localPath=localPath,
+                    serverURL=serverURL)
> x = configure(x, 'conf/description.txt', 'conf/plateconf.txt',
+                  'conf/screenlog.txt')
> x = annotate(x, 'conf/annotation.txt')
```

3.2 Inspecting data

We enumerate the non-empty wells with `getUnames` and retrieve metadata about them using `getWellFeatures`. The `controlStatus` field contains the well type. We then load the well phenotypic profiles using `readHTS` in the variable `xd`.

```
> us = setdiff(getUnames(x), getUnames(x, content='empty'))
> wfcontent = getWellFeatures(x, us)$controlStatus
> table(wfcontent)

wfcontent
 sample   empty     ubc    rluc   casp1 trappc3   clspn   kif11   kif23    plk1
   1558       0      24      24      24      24      24      24      24      24

> xd = readHTS(x, 'file', filename='data/profiles.tab', format='tab')
> xd = xd[match(us, xd$uname),]
```

There are 1750 non-empty wells in this screen, including 1558 sample experiments and 8 controls, each replicated 24 times. In the following example, we show how the median cell size `med.c.g.ss` and median cell eccentricity `med.c.g.ec` vary within well types.

```
> colors = c('#ffffff', NA, '#aaffff', '#ffaaff', '#ff44aa', '#aaaaff',
+            '#aaffaa', '#ff7777', '#aaaaaa', '#ffff77')
> par(mfrow=c(1,2))
> boxplot(xd$med.c.g.ss~wfcontent, las=2, col=colors,
+         main='Median cell size (a.u.)')
> boxplot(xd$med.c.g.ec~wfcontent, las=2, col=colors,
+         main='Median cell eccentricity (a.u.)')
```

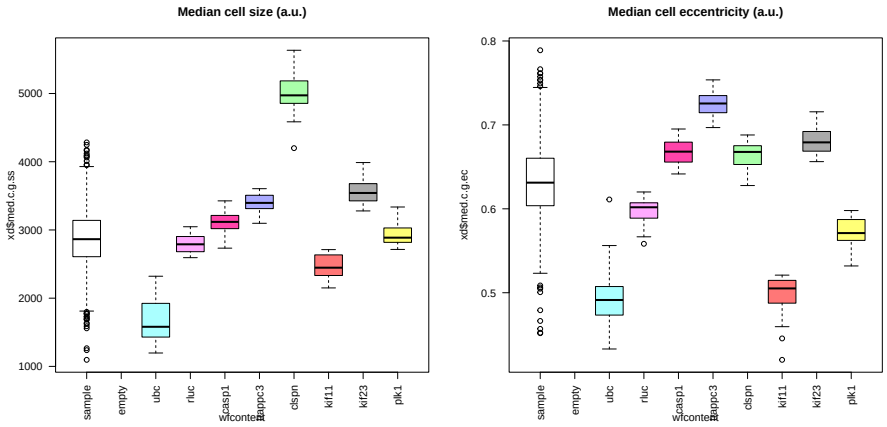

Figure 6: Distribution of median cell size and median cell eccentricity among well types

The boxplots show that the `ubc` control phenotype is characterized by small and round cells, the `clspn` control phenotype is characterized by large cells and the `trappc3` control phenotype is characterized by elongated cells.

To have a screen-wide overview of the well phenotypes, we draw in the following example a map of the phenotypic profiles using linear discriminant analysis (LDA), computed on the on the controls `rluc`, `ubc` and `trappc3`.

```
> library("MASS")
> z = wfcontent %in% c('rluc', 'ubc', 'trappc3')
> ft = 14:50
> ld = lda(xd[z, ft], as.character(wfcontent[z]))
> py = predict(ld, xd[, ft])
> plot(py$x[,1:2])
```

Two wells stand far away from the other ones. Are they novel phenotypes ? We identify and display them in the following example.

```
> unames = us[which(py$x[,1]>500)]
> print(unames)

[1] "001-01-A13" "002-01-I13"

> f = fileHTS(x, type='viewunmonted', spot=3, uname=unames[1])
> img1 = readImage(f)[1791:2238,1:448,]
> display(img1)
> f = fileHTS(x, type='viewunmonted', spot=1, uname=unames[2])
> img2 = readImage(f)[1:448,1:448,]
```

```
> display(img2)
```

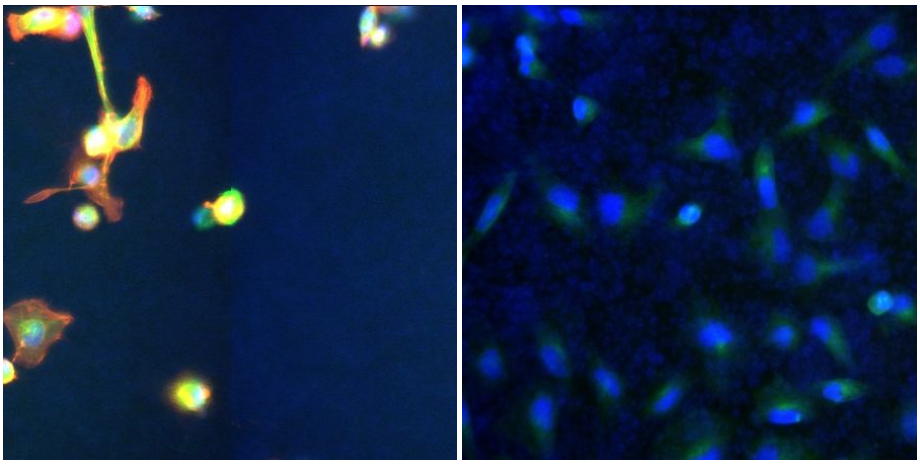

Figure 7: Well '001-01-A13' and '002-01-I13' showing staining problems

Wells '001-01-A13' and '002-01-I13' have serious staining problems. This is an example how a phenotypic map can be used for quality control. The wells cannot be used in the analysis and can be flagged in the screen log configuration file. The LDA plot is now redrawn by adjusting plot limits.

```
> plot(py$x[,1:2], xlim=c(-35,25), ylim=c(-20,20), cex=0.3)
> z = wfcontent!='sample'
> points(py$x[z,1:2], col=1, bg=colors[wfcontent[z]], pch=21)
> col = rep(1, length(levels(wfcontent)))
> col[2] = NA
> legend('topleft', legend=levels(wfcontent), col=col,
+        pt.bg=colors[1:length(wfcontent)], pch=21, ncol=2, cex=0.8)
```

Control wells `ubc`, `clspn`, `rluc` and `trappc3` are well separated from each other. Control wells `plk1` seem to display similar phenotypes than the negative control `rluc`: further inspection will reveal than the siRNA reagent against `plk1` did not work in this experiment.

Several sample wells seem to have strong phenotypes, distant from negative controls. Further data inspection is facilitated by the webQuery and cellPicker web modules, which allow interactive browsing and cell selection/annotation using a web browser. In the following example, the functions `popWebQuery` and `popCellPicker` open the corresponding modules. See Fig. 9 for an overview of the webQuery and cellPicker web modules.

```
> popWebQuery(x)
> uname = getUnames(x, content='trappc3')[1]
> popCellPicker(x, uname)
```

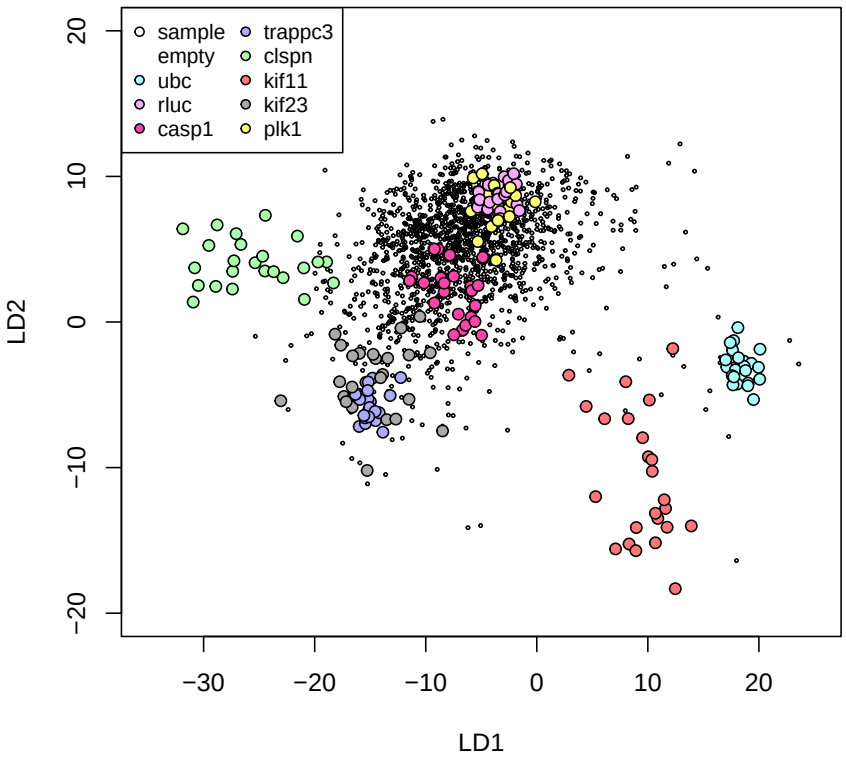

Figure 8: LDA projection of the phenotypic profiles, computed on the control `rluc`, `ubc` and `trappc3` wells

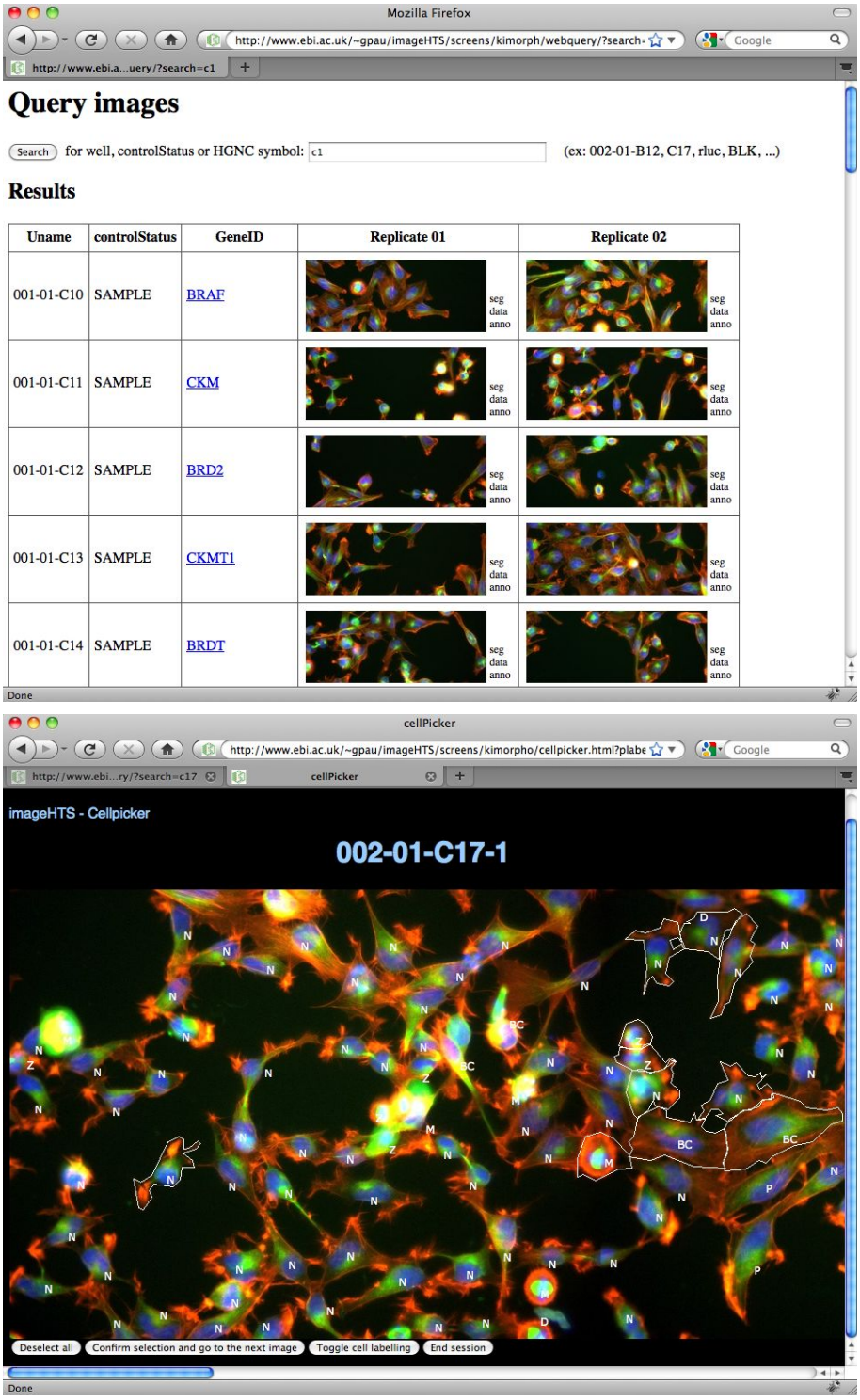

Figure 9: The webQuery (top) and cellPicker (bottom) web modules

4 Session info

This document was produced using:

- R version 4.1.2 (2021-11-01), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 20.04.3 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.14-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.14-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, grid, methods, stats, utils
- Other packages: Biobase 2.54.0, BiocGenerics 0.40.0, EBImage 4.36.0, MASS 7.3-55, RColorBrewer 1.1-2, cellHTS2 2.58.0, genefilter 1.76.0, hwriter 1.3.2, imageHTS 1.44.1, locfit 1.5-9.4, splots 1.60.0, vsn 3.62.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.56.2, BiocManager 1.30.16, BiocStyle 2.22.0, Biostrings 2.62.0, Category 2.60.0, DBI 1.1.2, GSEABase 1.56.0, GenomeInfoDb 1.30.0, GenomeInfoDbData 1.2.7, IRanges 2.28.0, KEGGREST 1.34.0, Matrix 1.4-0, R6 2.5.1, RBGL 1.70.0, RCurl 1.98-1.5, RSQLite 2.2.9, Rcpp 1.0.7, S4Vectors 0.32.3, XML 3.99-0.8, XVector 0.34.0, abind 1.4-5, affy 1.72.0, affyio 1.64.0, annotate 1.72.0, assertthat 0.2.1, bit 4.0.4, bit64 4.0.5, bitops 1.0-7, blob 1.2.2, cachem 1.0.6, class 7.3-20, colorspace 2.0-2, compiler 4.1.2, crayon 1.4.2, digest 0.6.29, dplyr 1.0.7, e1071 1.7-9, ellipsis 0.3.2, evaluate 0.14, fansi 1.0.0, fastmap 1.1.0, fftwtools 0.9-11, generics 0.1.1, ggplot2 3.3.5, glue 1.6.0, graph 1.72.0, gtable 0.3.0, htmltools 0.5.2, htmlwidgets 1.5.4, httr 1.4.2, jpeg 0.1-9, knitr 1.37, lattice 0.20-45, lifecycle 1.0.1, limma 3.50.0, magrittr 2.0.1, memoise 2.0.1, munsell 0.5.0, pillar 1.6.4, pkgconfig 2.0.3, png 0.1-7, preprocessCore 1.56.0, proxy 0.4-26, purrr 0.3.4, rlang 0.4.12, rmarkdown 2.11, scales 1.1.1, splines 4.1.2, stats4 4.1.2, survival 3.2-13, tibble 3.1.6, tidyselect 1.1.1, tiff 0.1-10, tools 4.1.2, utf8 1.2.2, vctrs 0.3.8, xfun 0.29, xtable 1.8-4, yaml 2.2.1, zlibbioc 1.40.0

References

[1] F. Fuchs, G. Pau, D. Kranz, O. Sklyar, C. Budjan, S. Steinbrink, T. Horn, A. Pedal, W. Huber, and M. Boutros. Clustering phenotype populations by genome-wide RNAi and multiparametric imaging. *Mol. Syst. Biol.*, 6:370, Jun 2010.

[2] M. Boutros, L. P. Bras, and W. Huber. Analysis of cell-based RNAi screens. *Genome Biol.*, 7:R66, 2006.

[3] G. Pau, F. Fuchs, O. Sklyar, M. Boutros, and W. Huber. EBImage–an R package for image processing with applications to cellular phenotypes. *Bioinformatics*, 26:979–981, Apr 2010.