`biopixR` - Tools for Biological Image Processing and Analysis

Tim Brauckhoff and Stefan Rödiger

2024-07-15

Contents

1 Introduction to Techniques in Bioimage Informatics for Feature Extraction

The volume of image data has increased rapidly due to the advancement of imaging technologies, including microscopy, confocal microscopy, and super-resolution techniques (H. Peng 2008; Swedlow, Goldberg, and Eliceiri 2009; Eliceiri et al. 2012; Sydor et al. 2015), as well as improvements in cell and tissue staining (Swedlow and Eliceiri 2009; Moen et al. 2019). These advancements have broad applicability in fields such as protein localization (Rigo et al. 2015), the environmental and cellular effects of microplastics (Cao et al. 2023; Jiang et al. 2024), diagnostics through microbead-assays (Dinter et al. 2023), deoxyribonucleic acid (DNA) damage assessment (Reddig et al. 2018; Schneider et al. 2019), and general cell biology (Ecke et al. 2019). The exponential growth in image data has rendered manual processing impractical, thereby risking accuracy and reproducibility (Caicedo et al. 2017). Consequently, the implementation of automated image data processing is of paramount importance in order to guarantee the objectivity and reproducibility of the results.

In light of recent developments, the utilization of microscopy in biomedical research has undergone a transformation, shifting from a predominantly visual approach to a quantitative one (Paul-Gilloteaux 2023). The demand for quantitative information from images to understand and develop biological concepts has led to the emergence of bioimage informatics as a specialized field (Eliceiri et al. 2012; Murphy 2014). Bioimage informatics is a field of study that focuses on the extraction of quantitative data from images with the aim of interpreting or developing biological concepts. The objective is to automate and objectively analyze image data while creating tools for visualization, storage, processing, and analysis (Swedlow, Goldberg, and Eliceiri 2009; H. Peng et al. 2012; Chessel 2017; Moen et al. 2019; Schneider et al. 2019). Achieving reproducible results, defined as consistent outcomes across experiments or studies conducted under similar conditions, is a primary objective in this field, with dedicated software playing a crucial role. Bioimage informatics employs computational methods to efficiently analyze large volumes of image data, encompassing key aspects such as image processing, machine learning, data management, and quantitative analysis (Schneider et al. 2019).

Fundamental operations in bioimage informatics include feature extraction, segmentation, registration, clustering, classification, annotation, and visualization (H. Peng 2008; Brauckhoff and Rödiger to be published). One of the principal techniques employed in the extraction of features from images is image segmentation, which is a prerequisite for subsequent quantification. It involves the division of an image into distinct Regions of Interest (ROI) by the assignment of labels to each pixel. The primary objective is to identify ROIs pertinent to the specific task (H. Peng 2008; Ghosh et al. 2019; Niedballa et al. 2022). Thresholding is a straightforward segmentation method. This approach involves comparing pixel values against one or more intensity thresholds, which results in the image being partitioned into foreground and background regions

(Sonka and Fitzpatrick 2000; Jähne 2002). Another common approach is the use of edge detection algorithms to outline objects of interest within an image (Canny 1986; Mittal et al. 2019). These techniques permit researchers to identify specific features within an image that may not be apparent through traditional manual analysis in a fast, reliable, and reproducible manner. The use of automated software ensures the consistency, reproducibility, and objectivity of the results obtained.

As described in a previous study, a multitude of software applications exist for the analysis of image data. (Schneider et al. 2019). In addition to Python, the statistical programming language `R` (R Core Team 2024) has become a central tool for data science and bioinformatics (Rödiger et al. 2015). As described in subsequent sections, a multitude of `R` packages have been developed with the specific purpose of performing image processing tasks. These packages address a range of requirements pertinent to bioimage informatics, including the importation, segmentation, and annotation of images. While some of these techniques will be discussed in the following sections, it is important to note that the existing packages do not cover all aspects. The subsequent sections will elucidate how the `biopixR` package contributes to the open-source image processing community by offering tools for feature extraction and automation. One significant application of the `biopixR` package is the analysis of round, spherical objects in images, such as microbeads, cells, seeds, or microplastics, which exhibit similar characteristics in their visual representations. Consequently, this vignette will predominantly feature examples from this domain.

2 History, Philosophy, and Aims of the `biopixR` Package

In 2018, we initiated the development of algorithms within the `R` programming language as part of our research in bioimage informatics. The primary objective was the analysis of data derived from microbead-based assays (for the quantification of nucleic acid and protein biomarkers) and cell-based assays (such as the analysis of DNA damage). During this period, we developed numerous scripts tailored to internal research projects and contributed to private repositories, including codeberg.org. It is noteworthy that our initial endeavors did not fully adhere to established software engineering practices, including unit testing, version tagging, and continuous integration (CI). For this reason they were never public.

As the algorithms we developed proved to be inefficient and inadequate for meeting current scientific needs, we resolved in 2023 to undertake a complete rewrite of the software. In October 2023, the initiative gained significant traction with a transition to open repositories on GitHub (https://github.com). The adoption of contemporary methodologies was intended to ensure enhanced software quality and facilitate greater collaboration with both the scientific and open-source communities. This had the immediate consequence that we received bug reports, stars, watches and contributions from other authors and users.

Our primary objective was to publish the package on the **C**omprehensive `R` **A**rchive **N**etwork (CRAN), which mandates high-quality software. This objective has been successfully achieved in 2024. Since then, multiple versions of the package have been released, including the initial milestone release (0.2.4) on April 2, 2024, and the subsequent stable version (1.0) on June 3, 2024. Since our initial contributions to CRAN, we have received valuable feedback and contributions from other package authors, including the author of the `data.table` package.

The `biopixR` package was initially employed for the analysis of microbeads (Geithe et al. 2021), and was subsequently utilized to perform quality control on microbeads in a 2024 study (Geithe et al. 2024).

The `biopixR` package has also been utilized in a recent publication (Dinter et al. 2023) for the precise quantification of signal intensities. This study aimed to develop novel hydrophobic microbeads for the precise quantification of amphiphilic molecules, such as phospholipids, on surfaces. These molecules are crucial in the development of a multitude of pathological conditions, including atherosclerosis, cardiovascular disease, infections, inflammatory disorders, cancer, and autoimmune diseases (Dinter et al. 2023).

The applications of `biopixR` can be extended to any research problem involving feature extraction from images and the quantification of related image data. Such envisioned applications include the assessment of wastewater for the detection of microplastics (Ding et al. 2020), the real-time localization of microbead-based drug delivery systems (Bannerman and Wan 2016), and other fields within the life sciences, such as cell biology (Schneider et al. 2019).

The aims of the `biopixR` package are to provide the functions needed for comprehensive image processing like:

- Convenient import of images in widely used formats.
- Tools for preprocessing images with highly fragmented contours.
- Versatile image processing functions for quantitative analysis.
- Interactive approaches to feature extraction.
- Integration of these functions to create user-friendly pipelines.
- Enabling batch processing and automation for medium-throughput analysis.

All technical and experimental aspects of `biopixR` are aimed to adhere to the principles of reproducible research. The development process was guided by the work of Wickham (2023) and the Guidelines provided by the R Core Team (2024). Encompassing the package building, metastructure, licensing, testing, documentation and distribution of the software. In accordance with the principles of *Agile Software Development* and *Extreme Programming*, several practices were implemented with the objective of ensuring the delivery of high-quality software that meets the needs of both end users and developers. These practices include version control, literate programming, unit testing, and continuous integration (Lanubile et al. 2010; Myers 2012; Rödiger et al. 2015; Gregory 2021).

3 Concepts and Methods

The following chapters provide an insight into the principles and methods used in the development process of the `biopixR` package, covering

- literate programming
- unit testing
- CI
- version control

as recommended by R. D. Peng, Kross, and Anderson (2016) and Wickham (2023). The development workflow encompassed the following steps:

1. Developing accurate segmentation strategies for microbeads.
2. Creating filter functions to discard specific undesirable characteristics.
3. Developing preprocessing algorithms to enable segmentation for droplet-based experiments.
4. Employing unsupervised machine learning to extract useful information.
5. Optimizing existing functions and integrating them for batch processing.
6. Conducting unit tests for verification and validation.

In addition, the `biopixR` package provides a unique data set of microbead images and microbeads in water-oil emulsions. These images serve as straightforward examples to demonstrate the capabilities and applications of the `biopixR` package.

For the development and testing of the `biopixR` package, a Lenovo ThinkPad E15 Gen2 was utilized, featuring 16 GB of RAM, an 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz × 8 processor, and a NV137 / Mesa Intel® Xe Graphics (TGL GT2) graphics chip. The operating system (OS) employed was Ubuntu 22.04.3 LTS 64-bit. The `R` version used was 4.3.2, and the development environment was RStudio 2023.09.0+463 "Desert Sunflower" Release 2023-09-25.

3.1 License and Broader Open Source Context

`biopixR` is an open-source software package (licensed under the GNU Lesser General Public License (LGPL) ≥ 3)[1] for the statistical programming language `R`, which is widely used in statistics, bioinformatics, and data science. The core contributors of the `biopixR` package are listed in the `DESCRIPTION` file. `R`'s active community has developed numerous packages for a multitude of applications, which facilitate the development of customized workflows, including data import, preprocessing, analysis, post-processing, and visualization, within a reproducible environment (Rödiger et al. 2015; Giorgi, Ceraolo, and Mercatelli 2022). The growing significance of image acquisition, processing, segmentation, feature extraction, and visualization in biological research underscores the importance of comprehensive data processing and automation capabilities. Although initially designed for statistical analysis, `R`, with its associated packages, is capable of effectively supporting image analysis and automation (Chessel 2017; Haase et al. 2022).

3.2 Version Control and Continuous Integration

For the purpose of version control, the widely used Git system, which is available on all major development platforms, was employed. Version control with Git enables the revision of changes and older versions of the code by providing complete repository copies. Additionally, it permits individual adaptation by creating distinct branches for the purpose of working on and experimenting with different versions while maintaining a stable one. Most importantly, it facilitates the organized sharing and merging of changes among team members, thereby significantly enhancing collaboration (Lanubile et al. 2010; Blischak, Davenport, and Wilson 2016; Vuorre and Curley 2018).

GitHub, a Git repository hosting provider, offers a web-based user interface to facilitate collaboration in open source projects. It incorporates tools for the reporting of bugs (Issues), collaboration (Pull requests),

[1] https://www.gnu.org/licenses/lgpl-3.0.de.html, accessed 07/11/2024

and workflows (Actions) (Spinellis 2012; Cosentino, Luis, and Cabot 2016; Perez-Riverol et al. 2016). The source code of the `biopixR` package is accessible at:

https://github.com/Brauckhoff/biopixR

CI is widely regarded as a good practice in software development. As team members frequently integrate their code, sometimes multiple times a day, the combination of code from different contributors can lead to significant issues with the software's integrity and functionality. To address this issue, CI is employed as an automated build and test system. It verifies the package's functionality and compatibility across various OS. This ensures that the code, package structure, metadata, and format remain functional. Therefore, CI is able to simplify the process of error detection by identifying potential issues directly within the integration process (Meyer 2014; Soares et al. 2022).

For `R`, the standard test suite is the `R CMD check`, which includes over 50 individual checks. These tests encompass a range of topics, including metadata validation, package structure, `DESCRIPTION` files, Namespace, `R` code, and documentation.[2] The `R CMD check` workflow for the `biopixR` package, based on the work of Hester (2021), involves testing across all major OS. The tests were conducted on Windows, macOS, and Linux. Furthermore, the developer version of `R` was tested on Linux. The source code for the CI setup using GitHub workflows, as well as the test history can be accessed at:

https://github.com/Brauckhoff/biopixR/actions/workflows/R-CMD-check.yml

3.3 Naming Convention and Literate Programming

`biopixR` is an `R` package ($\geq$ 4.2.0), designed using the S3 object system. S3 incorporates object-oriented programming features while simplifying development through naming conventions (Chambers 2014). Typically, functions and parameters in `R` packages are written using underscore separation (Bååth 2012). However, for the purpose of differentiation, this convention was adapted. Underscore separation is employed solely for variables and parameters introduced within the package. In accordance with the nomenclature convention proposed by Bååth (2012), the functions of the `biopixR` package adhere to the **lowerCamelCase** style (e.g., `objectDetection()`), with the exception to those designated to be interactive, which also utilize the **underscore_separated** style (e.g., `interactive_objectDetection()`).

To enhance the formatting, consistency, and readability of the code, the `styler` package by Müller and Walthert (2017) was employed and applied to the code. The `styler` package performs "non-invasive pretty printing of `R` code", whereby the code is formatted according to the *tidyverse style guide* (https://style.tidyverse.org/).

Literate programming, introduced by Knuth (1984), combines source code and documentation in a single file. This approach uses markup conventions (e.g., '#') to format the documentation, generating outputs in typesetting languages like **Markdown**. Literate programming is crucial for ensuring reproducibility of analysis in software development (Vassilev et al. 2016). Additionally, inline code annotations have been added to every function in the `biopixR` package.

The `roxygen2`, `rmarkdown`, and `knitr` packages were employed to write the documentation inline with the code for the `biopixR` package.

[2] https://r-pkgs.org/r-cmd-check.html, accessed 07/08/2024

3.4 Unit Testing of the `biopixR` Package

Software testing is a fundamental technique for the verification and validation of software, demonstrating the absence of errors. Module or unit testing is one such testing procedure, whereby individual subprograms, routines, or in `R`, functions are tested independently. This approach breaks down the entire package into smaller, more manageable components, rather than testing the whole software at once. A principal benefit of unit testing is the reduction of the debugging search area, as the specific function causing an issue is identified during testing (Myers 2012). Given that `R` is a package-based programming ecosystem, ensuring the correctness of distributed code is vital to guarantee the functionality of dependent packages (Vidoni 2021). A quantitative measure of the number of statements in a given code or function that are executed without error by a set of tests is described by the term *coverage* (Zhu, Hall, and May 1997; Vidoni 2021).

The objective of testing is to verify that specific inputs are processed correctly to generate the expected outputs and to ensure that error and warning statements operate as intended. In conclusion, tests confirm that the code performs as expected, and these expectations are recorded in reproducible scripts. In `R`, packages such as `RUnit`, `svUnit`, and `testthat` facilitate these tests (Wickham 2011; Myers 2012).

Unit tests for the `biopixR` package were created using the `testthat` package by Wickham (2009). The tests are executed automatically as part of the package building process and during the `R CMD check`. The unit test corresponding to each function is located in the `/tests/testthat/` subdirectory of the `biopixR` package. The following example provides insight into the testing procedure for the `changePixelColor()` function, with the expectation that:

- The function throws an error when importing an object that is not a 'cimg'.
- It does not throw an error when the input is correct: a 'cimg' object and coordinates as an x|y data frame.
- The `add.colour()` function (incorporated in `changePixelColor()`) transforms a grayscale image into one with three color channels.
- Normalization of `col2rgb()` results in values between 0 and 1 across three different channels.
- The color code is `0 0 1` for a pixel colored blue using the `changePixelColor()` function.
- The color code is `1 1 1` for a white pixel.
- The color code is `0 0 0` for a black pixel.

```
# Expectations and examples used for the unit testing of the 'biopixR' package
library(testthat)
library(biopixR)

test_that("changePixelColor", {
  mat <- matrix(0, 4, 4)
  mat[2:3, 2:3] <- 1
  img <- as.cimg(mat)
  coordinates <- data.frame(x = c(1, 3),
                            y = c(1, 3))

  expect_error(changePixelColor(mat, coordinates),
```

```
               regexp = "image must be of class 'cimg'")
  expect_no_error(changePixelColor(img, coordinates))

  expect_equal(dim(img)[4], 1)
  expect_equal(dim(add.colour(img))[4], 3)

  expect_equal(as.vector(col2rgb("red") / 255), as.vector(c(1, 0, 0)))
  expect_equal(as.vector(col2rgb("green") / 255), as.vector(c(0, 1, 0)))
  expect_equal(as.vector(col2rgb("blue") / 255), as.vector(c(0, 0, 1)))

  test <- changePixelColor(img, coordinates, color = "blue")
  expect_equal(test[1, 1, , ], as.vector(c(0, 0, 1)))
  expect_equal(test[2, 2, , ], as.vector(c(1, 1, 1)))
  expect_equal(test[1, 2, , ], as.vector(c(0, 0, 0)))
})
```

3.5 Installation of the `biopixR` Package

The ongoing developments will be consistently updated in the GitHub repository. Consequently, the latest developer version of the `biopixR` package can be accessed and downloaded directly from the repository using the `devtools` package.

```
# Install the 'devtools' package from CRAN.
# 'devtools' is required for installing R packages directly from GitHub repositories.
install.packages("devtools")

# Install the 'biopixR' package from a GitHub repository.
# 'install_github' is a function in 'devtools' that is used to install R packages
# hosted on GitHub.
# The argument "Brauckhoff/biopixR" specifies the GitHub username/repo of the package.
devtools::install_github("Brauckhoff/biopixR")
```

The `biopixR` package is available on CRAN, which can be accessed at:

https://CRAN.R-project.org/package=biopixR

CRAN employs rigorous testing procedures to ensure that the package can be downloaded and built on all major OS. Additionally, it validates the examples and documentation through the `R CMD check`. To utilize the `biopixR` package, it is first necessary to install `R` (version 4.2.0 or higher) and then to execute the following code:

```
# Install the 'biopixR' package from CRAN.
install.packages("biopixR")
```

The results of the `R CMD check` conducted by CRAN can be accessed via the following link:

https://cran.r-project.org/web/checks/check_results_biopixR.html

4 Functions for Quantitative Data Analysis in `biopixR`

An overview of the functions present in the `biopixR` package is provided in Figure 1, as the function in relation to the `imgPipe()` function are displayed.

```
foodweb(biopixR::imgPipe) |> plot()
```

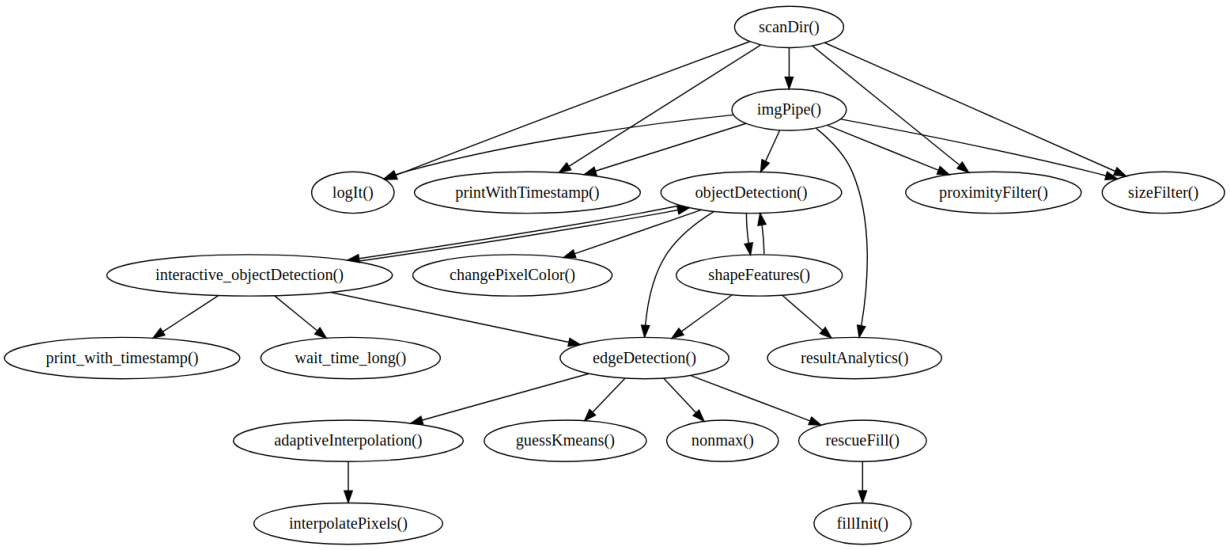

Figure 1: **Dependency Graph of Functions in the `biopixR` Package**: This graph illustrates the levels of complexity by depicting the descendants and ancestors of the `imgPipe()` function. The figure was created using the `foodwebr` package (version 0.1.1) from Appleton-Fox (2022), with RStudio 2023.09.0+463 and `R` 4.3.2 on Linux (Ubuntu 22.04.3 LTS).

The `biopixR` package includes a series of microbead images to demonstrate its capabilities in the analysis and processing of biological images (Figure 2). For further information regarding microbead assays and their utilisation in biomedical research, please refer to the following reference (Rödiger et al. 2014).

The sample images illustrate the package's functionalities, allowing users to explore and experiment with image analysis and manipulation within the context of biotechnology and life sciences. Researchers and practitioners may utilize these illustrations to comprehend the applicability of `biopixR` to their particular imaging requirements, whether pertaining to cell biology, microscopy, or other biological imaging applications.

```
# Load the biopixR package
library(biopixR)
```

```r
# Set up a 2x2 plotting area
par(mfrow = c(2, 2))

# Plot example images without axes and with a title
plot(beads, axes = FALSE, main = "beads")
plot(beads_large1, axes = FALSE, main = "beads_large1")
plot(beads_large2, axes = FALSE, main = "beads_large2")
plot(droplet_beads, axes = FALSE, main = "droplet_beads")

# Reset the plotting area to a single plot
par(mfrow = c(1, 1))
```

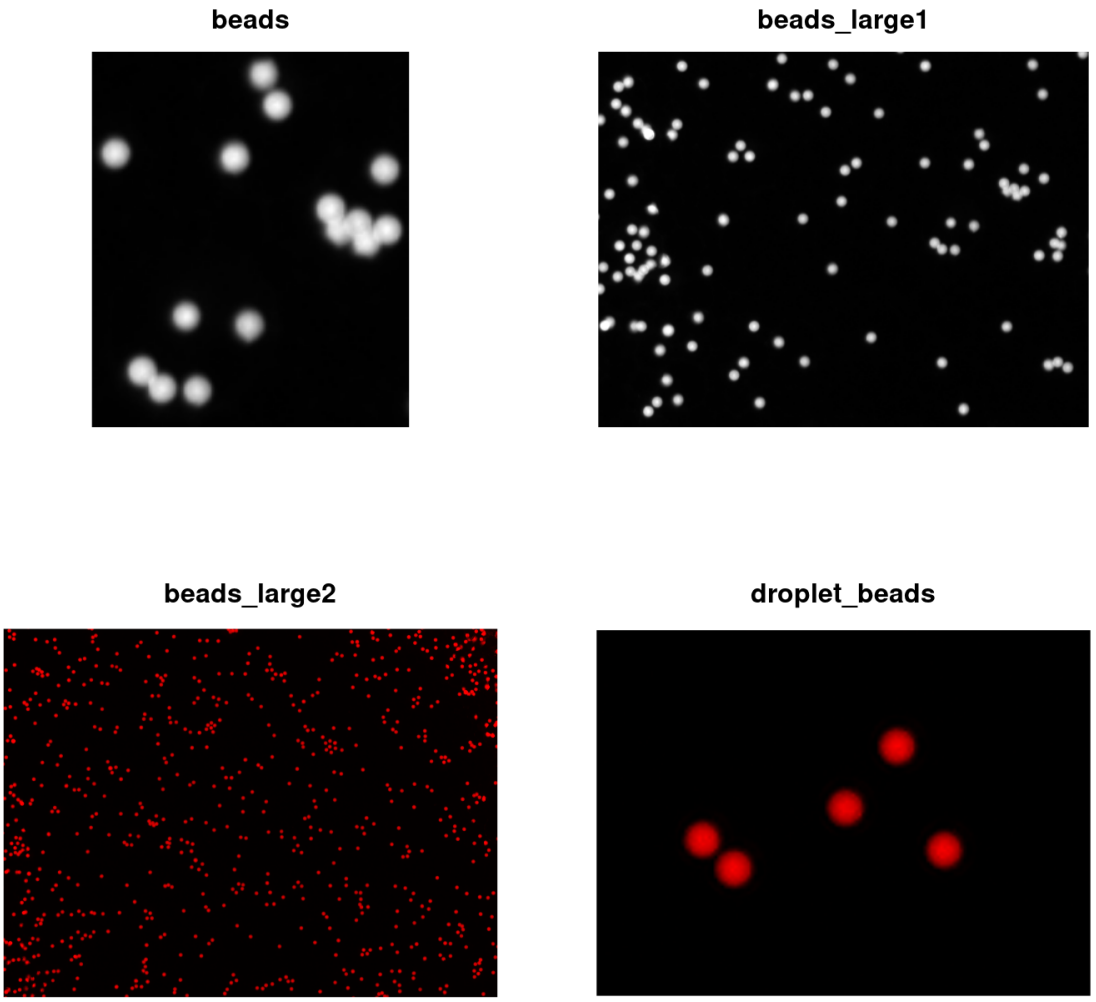

Figure 2: **Examples**: Images of microbeads provided by the `biopixR` package.

A selection of these microbead images will be employed in the forthcoming demonstration of the `biopixR`

functions. As previously mentioned, the primary development objective was to analyze data derived from microbead-based and cell-based assays.

This vignette will focus on the analysis of microbead particles made of polymethylmethacrylate (PMMA), which are approximately 12 µm in size (Geithe et al. 2024). **In the following sections and illustrations, there is no further mention of the size of microbeads or any specifications provided in the images. All algorithms are indifferent to diameter.** At the end of this vignette, you will find a brief example for analyzing cell-based assays.

4.1 `importImage()` - Importing Images into the `R` Environment

The `biopixR` package features an import function called `importImage()`. The function supports the importation of digital images in various file formats, including Joint Photographic Experts Group (JPEG), Portable Network Graphics (PNG), Bitmap Image File (BMP), and Tagged Information Interchange Format (TIFF). This function acts as a wrapper, integrating the capabilities of the `magick` and `imager` packages for import and class conversion. Since most image processing operations in `biopixR` rely on `imager`, the `importImage()` function converts all formats into the `imager` class 'cimg'. During the development process, it was frequently observed that images exhibited more than three dimensions within the color channel, specifically an additional transparency layer, also known as alpha. Such images often lead to challenging and elusive errors. To address this issue, the `importImage()` function employs a process of detection and removal of the fourth color dimension, if present.

The import function is demonstrated in Figure 3, where it is used to import two microbead images in BMP and PNG formats.

```
# Get the path to the 'beads.png' image file within the 'biopixR' package
path2img <- system.file("images/beads.png", package = "biopixR")

# Import the image from the path specified by 'path2img' and store it in the
# 'microbeads' object
microbeads <- importImage(path2img)

# Import the image 'fig6.1_transparent.bmp' from the 'figures' directory and
# store it in the 'transparent_bead' object
transparent_bead <- importImage("figures/fig6.1_transparent.bmp")

# Display the class of the 'microbeads' object
class(microbeads)
```

```
[1] "cimg"         "imager_array" "numeric"
```

```
# Set up a 1x2 plotting area
par(mfrow = c(1, 2))
```

```r
# Display imported images
plot(microbeads, axes = FALSE)
plot(transparent_bead, axes = FALSE)

# Reset the plotting area to a single plot
par(mfrow = c(1, 1))
```

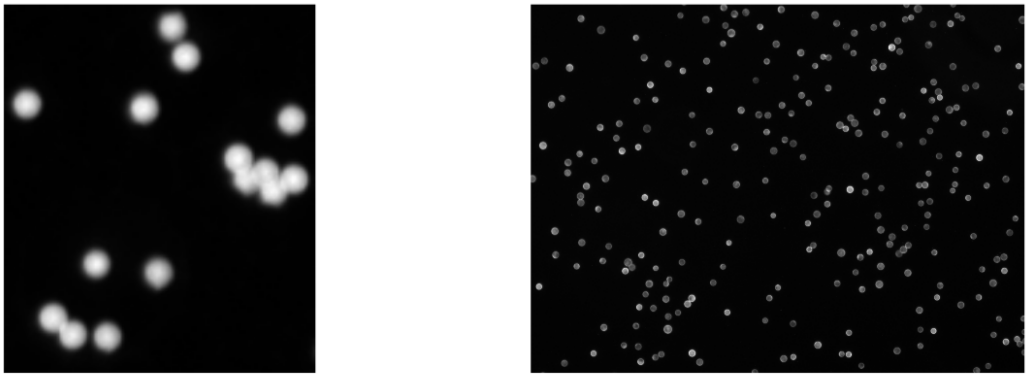

Figure 3: **Example Images**: Showcasing the functionality of the function for image import - `importImages()`, with two images of microbeads.

4.2 `edgeDetection()` - A modified Canny Edge Detector

Edge detection (e.g., contours, lines) is crucial in various applications such as computer vision and object recognition, particularly in medical image analysis. It aids in identifying regions with significant changes or transitions within an image, thereby extracting valuable information by highlighting key elements like shapes, patterns, or specific areas of interest. Various edge detection algorithms are available in `R`, with the Canny edge detector, developed by Canny (1986), being the most widely implemented across multiple packages, including those by Barthelme (2015) (`imager`); Ooms (2016) (`magick`); Mouselimis (2016) (`OpenImageR`); and Beare, Lowekamp, and Yaniv (2018) (`SimpleITK`). Other noteworthy edge detection algorithms in the `R` package `OpenImageR` includes those by Prewitt et al. (1970), Sobel (2014), Roberts (1980), and Scharr (2000). After evaluating the different results (e.g., Figure 4), the Canny edge detection algorithm from the `imager` package was selected for edge detection. The resulting binary image serves as a foundation for subsequent feature extraction. Furthermore, the `cannyEdges()` function in `imager` offers adjustable parameters for `alpha` and `sigma`, enabling users to customize thresholding and smoothing, providing the desired flexibility.

The process of Canny edge detection using the `imager` package comprises a series of steps. Initially, a Gaussian filter is applied to the image, resulting in the smoothing of the image to remove noise. The degree of smoothing can be adjusted by varying the value of the `sigma` parameter. Subsequently, the intensity gradient is calculated to determine the magnitude of the edges. This is followed by the application of non-maximum

suppression, which serves to minimize the blur introduced previously. Subsequently, a double threshold is applied. In the absence of provided thresholds, they are estimated through *k-means* clustering. The calculated threshold can be adjusted using the `alpha` parameter. These thresholds are employed to classify edges as either weak or strong. Finally, hysteresis is employed to combine these edges, with weak edges being discarded if they are not in proximity to strong edges (Barthelme 2015; Barthelmé and Tschumperlé 2019).[3]

```
# Set up a 1x2 plotting area
par(mfrow = c(1, 2))

# Edge detection with 'Prewitt' method
OpenImageR::edge_detection(as.matrix(beads),
                           method = "Prewitt") |>
  as.cimg() |>
  plot(axes = FALSE,
       main = "Prewitt - edge detection",
       cex.main = 3.5)
text(c(10), c(10), c("A"), col = "darkred", cex = 3.5)

# Edge detection with 'Canny' method
cannyEdges(beads) |> plot(axes = FALSE,
                          main = "Canny - edge detection",
                          cex.main = 3.5)
text(c(10), c(10), c("B"), col = "darkred", cex = 3.5)

# Reset the plotting area to a single plot
par(mfrow = c(1, 1))
```

[3] http://dahtah.github.io/imager/canny.html, accessed 06/26/2024

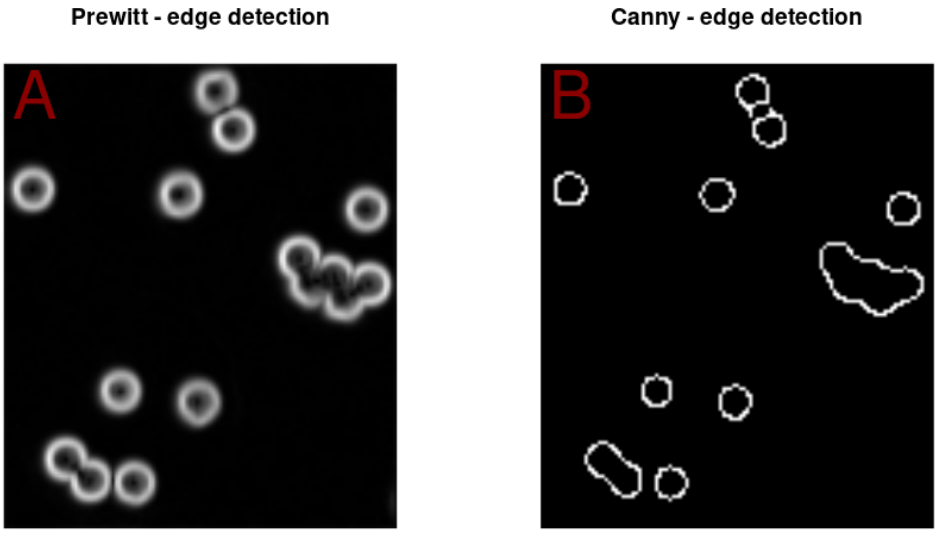

Figure 4: **Comparison of Two Edge Detection Algorithms**: **A**) Resulting image using the Prewitt et al. (1970) edge detection algorithm from the `OpenImageR` package. The contours have higher intensity, but the image is not binary. **B**) The Canny edge detection algorithm produces a binary image with distinct contours.

In the context of microbead images, the contours that were identified frequently exhibited gaps, rendering them inaccessible for subsequent labeling (Figure 5A). The figure depicts filled circles representing successfully labeled microbeads, while the contours indicate unsuccessful labeling. Consequently, segmentation with fragmented contours is incomplete, rendering the objects inaccessible for further analysis. To address this issue, the `magick` package was employed to identify line ends (Figure 5B). Line ends can then be reconnected to a neighboring line, provided that the line end does not share the same label. This process of reconnection is constrained by a specific radius to prevent line ends from connecting across the entire image.

```
# Set up the plotting area to have 1 row and 2 columns
par(mfrow = c(1, 2))

# Apply Canny edge detection to the image 'beads_large1' with specified parameters
edge_canny <- cannyEdges(beads_large1, alpha = 0.8, sigma = 0)

# Label the detected edges
labeled_canny <- label(edge_canny)

# Plot the labeled edges without axes
plot(labeled_canny, axes = FALSE)
text(c(475), c(355), c("A"), col = "darkred", cex = 5)
```

```r
# Draw red arrows at specified coordinates
arrows(
  x0 = 23,
  y0 = 29,
  x1 = 24,
  y1 = 30,
  col = "red",
  lwd = 3
)
arrows(
  x0 = 412,
  y0 = 148,
  x1 = 413,
  y1 = 147,
  col = "red",
  lwd = 3
)
arrows(
  x0 = 73,
  y0 = 210,
  x1 = 72,
  y1 = 210,
  col = "red",
  lwd = 3
)

# Mirror the detected edges across the x-axis
edge_canny_m <- mirror(edge_canny, axis = "x")

# Convert the mirrored edge image to magick format
canny_magick <- cimg2magick(edge_canny_m)



# Detect the coordinates of all line ends using morphology operation
lineends_canny <- image_morphology(canny_magick,
                                   "HitAndMiss", "LineEnds")

# Convert the extracted coordinates back into 'cimg' format
lineends_cimg <- magick2cimg(lineends_canny)

# Find the coordinates of the line ends and transform into a data frame
end_points <- which(lineends_cimg == TRUE, arr.ind = TRUE)
end_points_df <- as.data.frame(end_points)
```

```r
colnames(end_points_df) <- c("x", "y", "dim3", "dim4")

# Highlight the line end pixel in green color on the original edge image
endpoints_img <- changePixelColor(as.cimg(edge_canny),
                                  end_points_df,
                                  color = "green",
                                  visualize = FALSE)

# Plot the image with highlighted line ends without axes
plot(endpoints_img, axes = FALSE)
text(c(475), c(355), c("B"), col = "darkred", cex = 5)

# Highlight the line ends in green color on the original edge image
points(end_points_df$x, end_points_df$y, col = "green", lwd = 2)

# Draw red arrows at specified coordinates
arrows(
  x0 = 23,
  y0 = 29,
  x1 = 24,
  y1 = 30,
  col = "red",
  lwd = 3
)
arrows(
  x0 = 412,
  y0 = 148,
  x1 = 413,
  y1 = 147,
  col = "red",
  lwd = 3
)
arrows(
  x0 = 73,
  y0 = 210,
  x1 = 72,
  y1 = 210,
  col = "red",
  lwd = 3
)

# Set up the plotting area back to normal
par(mfrow = c(1, 1))
```

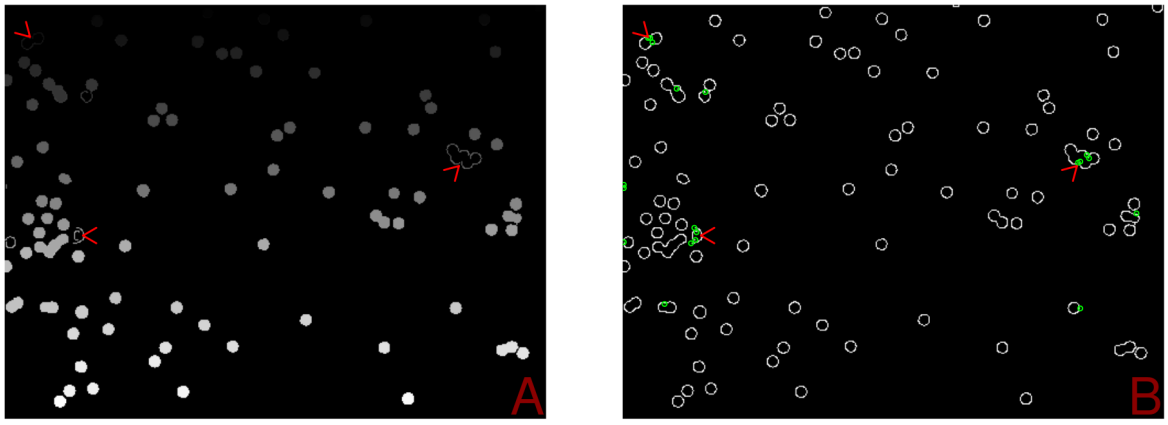

Figure 5: **Segmentation Result with Canny Edge Detector**: **A**) Segmentation result using the `label()` function. The segmentation is incomplete, as not all microbeads are identified as foreground. Only the contours are detected in these cases (highlighted by red arrows). **B**) Result of the `cannyEdges()` function, showing detected line end pixels, which are colored and circled in green.

As illustrated in Figure 6A, the modified Canny edge detector, `edgeDetection()`, is capable of successfully rejoining line ends, thereby enabling the detection of previously unlabeled microbeads, as shown in Figure 5A. For further visualization, the `objectDetection()` function was employed. This function employs the `edgeDetection()` function and provides visual feedback as an output (Figure 6B).

```
# Set up a 1x2 plotting area
par(mfrow = c(1, 2))

# Detect objects in the 'beads_large1' image using the edge method with
# specified alpha and sigma values
object_biopixR <-
  objectDetection(beads_large1,
                  method = 'edge',
                  alpha = 0.8,
                  sigma = 0)

# Perform edge detection on the 'beads_large1' image with specified alpha and
# sigma values
edge_biopixR <- edgeDetection(beads_large1, alpha = 0.8, sigma = 0)

# Label the detected edges in the 'edge_biopixR' image
```

```r
labeled_biopixR <- label(edge_biopixR)

# Plot the labeled edges without axes
plot(labeled_biopixR, axes = FALSE)
text(c(475), c(355), c("A"), col = "darkred", cex = 5)

# Plot the marked objects from the object detection without axes
plot(object_biopixR$marked_objects, axes = FALSE)
text(c(475), c(355), c("B"), col = "darkred", cex = 5)

# Reset the plotting area to a single plot
par(mfrow = c(1, 1))
```

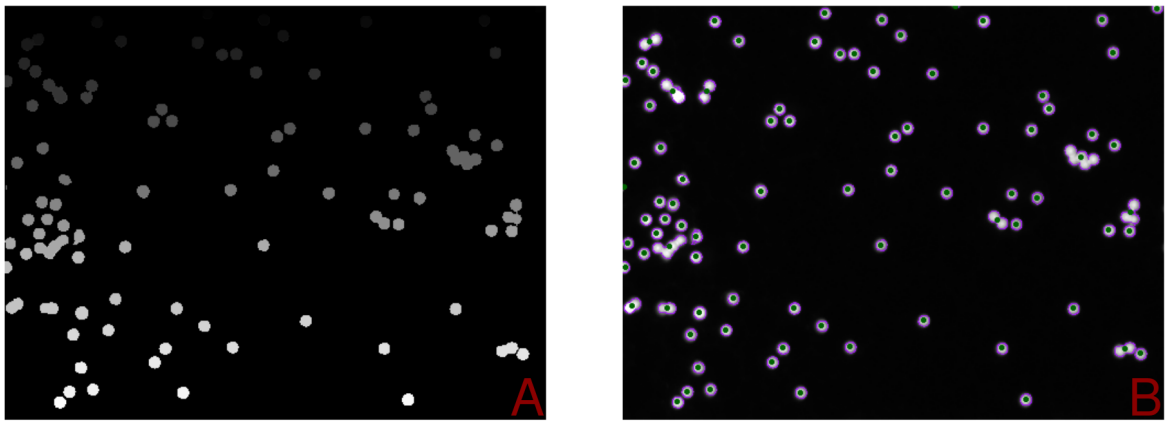

Figure 6: **Segmentation Result with Modified Canny Edge Detector**: **A**) Segmentation result using the `label()` function, showing successful segmentation with all microbeads identified as part of the foreground. **B**) Successful segmentation visualized using the `objectDetection()` function, with purple contours around each microbead and green dots indicating their centers.

4.3 `objectDetection()` - A Function for Feature Extraction

The `objectDetection()` function of the `biopixR` package serves as a segmentation tool, offering various methods for the extraction of objects of interest from an image. The extracted information includes the center, size, and coordinates of each detected object. In the case of microbeads, it is essential to distinguish between individual entities. Consequently, the `objectDetection()` function provides tools for segmentation

using either thresholding or edge detection. Both methods have distinct advantages depending on the specific application, and their respective use cases will be presented in the following sections.

4.3.1 Edge Detection Method in `objectDetection()`

When edge detection is selected, the modified Canny edge detector, provided by the `edgeDetection()` function, is used. As previously described in Chapter 4.2, the selection of the `alpha` and `sigma` parameters plays a pivotal role in the performance of feature extraction. The objective is to modify the threshold (`alpha`) to detect all objects without detecting noise, and to maintain a minimal level of smoothing (`sigma`) to avoid merging proximate objects and to ensure accurate edge detection. To facilitate the selection of parameters, the `biopixR` package offers a range of methods designed to assist users in this process:

Automated Parameter selection:

The `biopixR` package encompasses two distinct methods for the automated parameter selection process. Both automation methods employ a fitness function to extract shape information via the `shapeFeatures()` function. The fitness function evaluates the results using various input parameters, operating under the assumption that the objects in question are circular. While the grid search method is time-consuming due to its exhaustive testing of every possible parameter combination, the Pareto front optimization method samples and analyzes a subset of combinations, allowing for a more rapid estimation of the optimal parameters.

Grid Search - To conduct a grid search, it is first necessary to create a parameter grid containing all possible combinations of the `alpha` and `sigma` parameters. The range of `alpha` is predetermined and fixed at 0.1 to 1.5, while the range of `sigma` is also predetermined and fixed at 0 to 2, with both ranges incrementing by 0.1. This process yields a grid of 315 objects, which represent the potential parameter combinations to be tested. For each combination, a fitness value is calculated using the fitness function, with the entire parameter grid being scanned in sequence. To achieve a balance between the two objectives of attaining circular shaped objects and detecting all objects, the fitness value representing the shape is combined with the number of detected objects. The results of this method are presented in Figure 7. The image analysis process is relatively time-consuming, with a runtime of approximately five minutes for the example image. Additionally, four of the microbeads could not be successfully detected.

```
# Start the timer to measure the execution time of the code block
tictoc::tic()

# Perform automated object detection on the 'beads_large1' image
static_result <- objectDetection(beads_large1,
                                 method = 'edge',
                                 alpha = 'static',
                                 sigma = 'static')

# Stop the timer and display the elapsed time
tictoc::toc()
```

```
257.414 sec elapsed
```

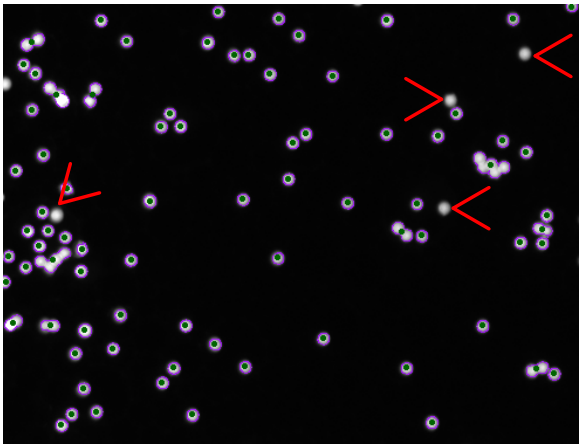

Figure 7: **Results of Automated Feature Extraction Using Grid Search**: The successfully detected microbeads are highlighted through the internal visualization of the `objectDetection()` function. The contours are outlined in purple, and the center of each object is marked with a green circle. Notably, four microbeads were not detected by the algorithm (indicated by the red arrow).

Multi-objective Optimization - As mentioned above, automation must strike a balance between detecting only objects with perfect shapes (losing information about the phenomena discussed above) and detecting noise or unwanted areas around the objects, which can lead to the merging of nearby objects as seen in thresholding (Figure 10). In grid search, combining both quality (shape features) and quantity (number of objects detected) measures into a single value while maintaining this balance is a significant challenge. These measures are controlled by the input parameters `alpha` and `sigma`. To solve this problem, another method specifically designed to optimize multiple parameters, known as multi-objective optimization, has been used.

In `R`, this method is accessible through the `GPareto` package, which is designed for 'Gaussian process-based multi-objective optimization'. This approach is particularly well-suited to computationally intensive optimization tasks (Binois and Picheny 2019). This criterion is met because the optimal parameter combination must be identified through the extraction of shape features for each parameter combination, which is a process that requires significant computational resources.

The objectives to be optimized are the parameters `alpha` and `sigma`. The Gaussian process is used to model the objective functions based on a limited number of sample points. Gaussian regression predicts unknown values by modeling the spatial correlation between sample points (Binois and Picheny 2019). The optimization task can be formulated as follows: to detect all objects while maintaining circular shape characteristics and avoiding the merging of nearby objects (quality/quantity trade-off).

The `GPareto` package aims to identify the set of optimal compromises, known as the Pareto set, consisting

of non-dominated points (points where no other point has better objectives). The visualization of the Pareto set in the objective space is called the Pareto front (Binois and Picheny 2019) (Figure 8).

The default criterion selected is SMS-(EGO) - S-metric Selection Efficient Global Optimization, an extension of the EGO algorithm tailored for multi-objective optimization. This method, used as an infill criterion, selects new sample points by maximizing hypervolume improvement, effectively balancing exploration (finding new areas) and exploitation (refining known good areas). Particle Swarm Optimization (PSO) is used as the internal optimization routine to find the optimal sampling point (Binois and Picheny 2019).

The principle of this method involves:

1. Generating an initial set of observations.
2. Fitting the Gaussian process models to each objective independently.
3. Running an inner optimization loop to find the best point (new point as the maximizer of an infill criterion).
4. Obtaining a new observation by running the simulator and updating the models accordingly.

The last two steps are repeated until the simulation budget of 20 is exhausted or a stopping criterion is met (Binois and Picheny 2019).

```
# Start the timer to measure the execution time of the code block
tictoc::tic()

# Perform automated object detection on the 'beads_large1' image
gaussian_result <- objectDetection(beads_large1,
                                   method = 'edge',
                                   alpha = 'gaussian',
                                   sigma = 'gaussian')
```

```
----------------------------
Starting optimization with :
 The criterion SMS
 The solver pso
----------------------------
Ite / Crit / New x / New y
No refPoint provided,  1.23 -8200 used
1   /   -1980   /   0.1 2   /   0.301   -8390
2   /   -3790   /   0.1 0   /   0.268   -9400
refPoint changed,  1.27 -8180 used
3   /   -1420   /   0.37    0   /   0.234   -9390
4   /   -542    /   0.525   0   /   0.235   -9390
5   /   -507    /   0.834   0.85    /   0.224   -9000
6   /   -935    /   0.1 0.795   /   0.244   -9080
7   /   -376    /   0.432   0.424   /   0.231   -9300
8   /   -153    /   0.271   0.287   /   0.244   -9370
```

```
9   /   -153    /   0.888   0.414   /   0.231   -9300
10  /   -80.8   /   0.295   1.34    /   0.238   -8600
11  /   -89.6   /   0.707   1.47    /   0.231   -8480
12  /   -84.3   /   0.718   0.194   /   0.235   -9390
```

```
# Stop the timer and display the elapsed time
tictoc::toc()
```

```
43.987 sec elapsed
```

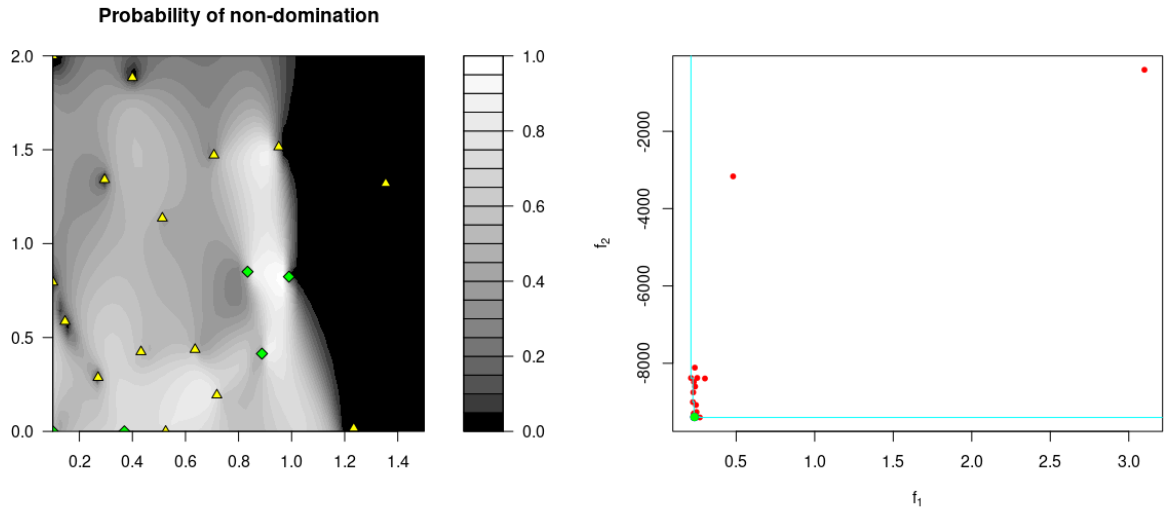

Figure 8: **Uncertainty Plot of the Objective Space** (left) & **Pareto Front Approximation** (right): The Uncertainty Plot provides a visualization of the confidence levels in the estimate of the Pareto front using Gaussian process modeling (x-axis: `alpha`; y-axis: `sigma`). The sample points used to construct the model are highlighted in yellow, while the actual realizations of the Pareto set are highlighted in green. In addition, the probability shades categorize regions of interest as white (indicating regions of high interest), black (indicating regions of low interest), and gray (indicating regions of uncertainty). The Pareto Front Plot visualizes the non-dominated solutions, known as the Pareto set. The optimal point within this set is highlighted in green. The input parameters `alpha` and `sigma` corresponding to this optimal point were used for the final analysis of the microbead image.

The optimal `alpha` and `sigma` parameters, obtained through Gaussian process modeling, are utilized as input for the `edgeDetection()` algorithm within the `objectDetection()` function. The detected microbeads are displayed in Figure 9, with all microbeads being successfully identified.

```
# Display the result of the automated parameter calculation using multi-objective
# optimization
plot(gaussian_result$marked_objects, axes = FALSE)
```

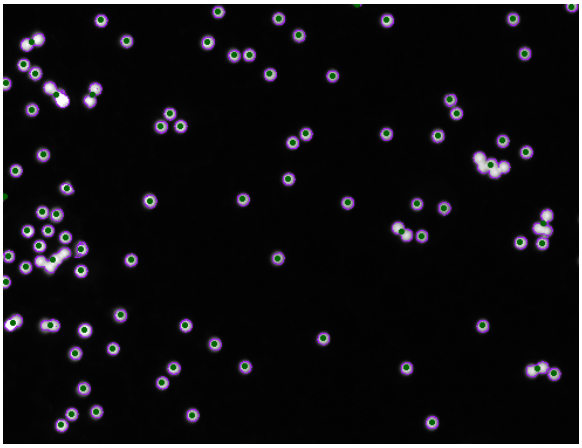

Figure 9: **Results of Automated Feature Extraction Using Multi-objective Optimization**: The detected microbeads are highlighted through the internal visualization of the `objectDetection()` function. The contours are outlined in purple, and the center of each object is marked with a green circle. Notably, all the microbeads are detected.

4.3.2 Thresholding Method in `objectDetection()`

Thresholding is a technique used in image processing to divide an image into various regions based on their brightness or intensity values. By setting a specific threshold, it is possible to distinguish between foreground objects (above the threshold) and background areas (below the threshold). This differs from edge detection, which focuses on identifying abrupt changes in pixel intensities within images to locate boundaries of objects or shapes. The thresholding method is particularly well-suited for images with high and inhomogeneous backgrounds, as it incorporates background correction by solving the Screened Poisson Equation (SPE) before applying the threshold. This correction is achieved through the use of the `SPE()` function from the `imagerExtra` package (Ochi 2018), which is based on the method described by Morel, Petro, and Sbert (2014). This approach addresses image artifacts such as inhomogeneous illumination and low contrast while preserving image details (Morel, Petro, and Sbert 2014). Consequently, the method allows for the detection of low-contrast objects against inconsistent backgrounds, such as transparent microbeads (Figure 10). The thresholding method does not require any additional input, is highly robust and also performs well on fluorescent microbead images. However, this approach has one disadvantage: the threshold is less strict than edge detection, which may result in the merging of objects in proximity that would be regarded as separate entities by the edge detection method.

```r
# Set up the plotting area to have 1 row and 2 columns
par(mfrow = c(1, 2))

# Import the image of transparent beads
transparant_beads <-
  importImage("figures/fig6_transparent_beads.bmp")

# Plot the imported image without axes
plot(transparant_beads, axes = FALSE)
text(c(70), c(70), c("A"), col = "darkred", cex = 5)

# Perform object detection on the transparent beads image using the 'threshold'
# method
result_transparant <-
  objectDetection(transparant_beads, method = 'threshold')

# Plot the marked objects from the object detection result without axes
plot(result_transparant$marked_objects, axes = FALSE)
text(c(70), c(70), c("B"), col = "darkred", cex = 5)

# Visualize merged microbeads
arrows(900, 800, 899, 801, col = "red", lwd = 7.5)

# Reset the plotting area to the default 1 row and 1 column
par(mfrow = c(1, 1))
```

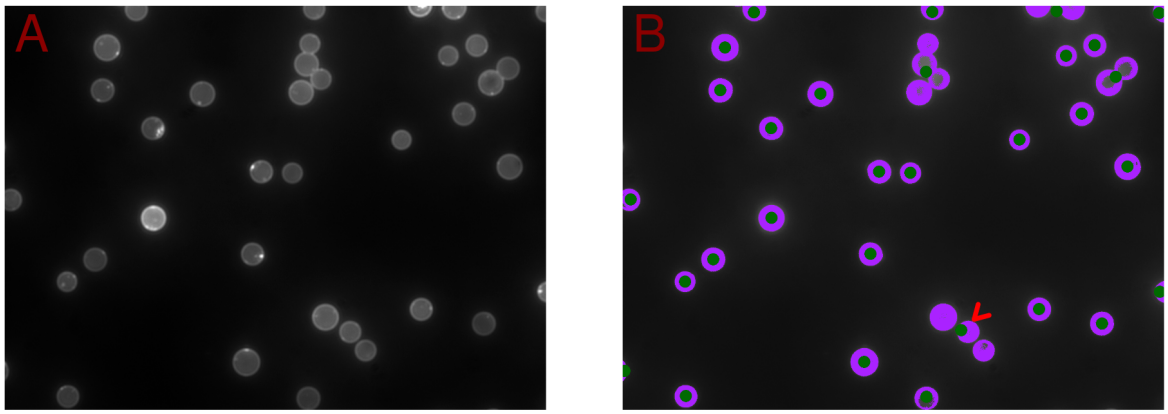

Figure 10: **Transparent Microbeads**: **A**) Original image of transparent microbeads, showing low contrast between the microbeads and the background. **B**) Application of the threshold method, resulting in the detection of the outer layer (halo) of the microbeads, successfully capturing the ligand signal. Detected coordinates are colored in purple, with centers marked in green. The merging of proximate microbeads into a single entity is exemplified by the red arrow.

4.3.3 Interactive Approach in `objectDetection()`

To facilitate parameter selection, an interactive object detection function was developed using the `R` package `tcltk`, which provides access to Tcl/Tk in `R`. The Graphical User Interface (GUI) is invoked via the `interactive_objectDetection()` function (Figure 11). The `alpha` and `sigma` parameters of the `edgeDetection()` function, discussed in Chapter 4.2, represent the threshold adjustment factor and the smoothing factor, respectively. The corresponding sliders in the Tcl/Tk interface enable users to adjust these parameters to optimize object detection. The detected objects are highlighted with purple contours and green centers. The "Switch Method" button enables the user to toggle between edge detection and the threshold method for object detection. Figure 11 illustrates that the threshold adjustment factor - `alpha` - must be decreased, as not all microbeads are currently being detected. To facilitate the analysis of smaller images, a scaling slider was incorporated into the interface, utilizing the `imresize()` function from the `imager` package. This function employs bilinear interpolation to adjust the image size according to user input (Barthelme 2015).

```
# Open interactive Tcl/Tk interface for object Detection
interactive_objectDetection(beads_large1)
```

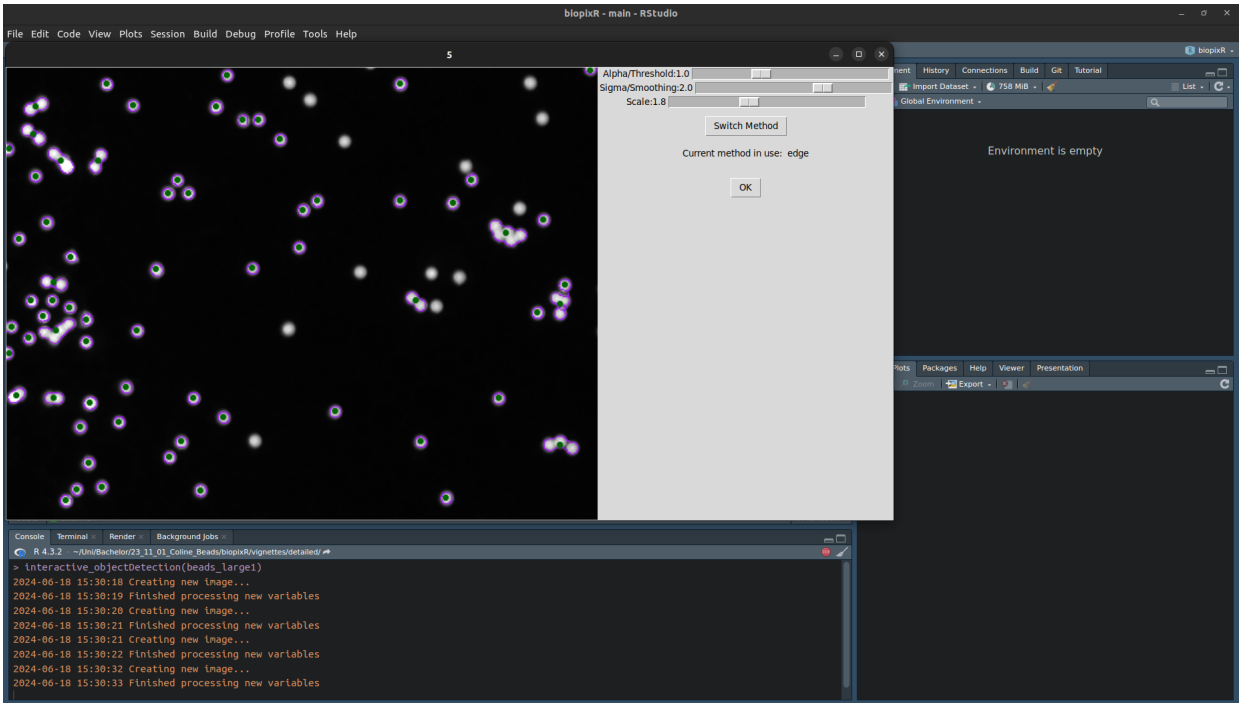

Figure 11: **Graphical User Interface for Interactive Parameter Selection**: The `interactive_objectDetection()` function offers a user-friendly interface with sliders to adjust threshold, smoothing, and scale. It also includes a button to switch between edge detection and thresholding methods. Object contours are highlighted in purple and centers in green for easy visualization. This example was executed in RStudio 2023.09.0+463 with `R` 4.3.2 on Linux (Ubuntu 22.04.3 LTS), displaying fluorescent microbeads. The rendering process, including timestamps and the current state, is shown in the console.

In conclusion, the `objectDetection()` function gathers comprehensive data about image objects, which facilitates the identification and differentiation of individual features. This process provides precise coordinates for each object in the image, which serve as the foundation for further analysis and characterization of features within the `biopixR` package. Moreover, the function provides a variety of methods for extracting objects within an image, thereby ensuring its adaptability for a broad range of applications.

4.4 Dealing with Fluorescence Reflection and Aggregation with `sizeFilter()` and `proximityFilter()`

In microbead-based assays, two phenomena must be avoided to prevent the generation of low-quality or incorrect results. These issues are exemplified in one of the images provided by the package (Figure 12). The first issue arises when microbeads are in proximity to one another. As laterally-emitted light (Göröcs, McLeod, and Ozcan 2015) has the potential to cause reflection from adjacent microbeads, potentially generating false positive signals. To address this issue, we developed the `proximityFilter()` function.

```
# Highlight unwanted microbead phenomena
plot(beads, axes = FALSE)
arrows(
```

```
  x0 = 60,
  y0 = 17,
  x1 = 61,
  y1 = 16,
  col = "green",
  lwd = 2
)
arrows(
  x0 = 90,
  y0 = 75,
  x1 = 91,
  y1 = 74,
  col = "cyan",
  lwd = 2
)
arrows(
  x0 = 30,
  y0 = 116,
  x1 = 29,
  y1 = 117,
  col = "red",
  lwd = 2
)
```

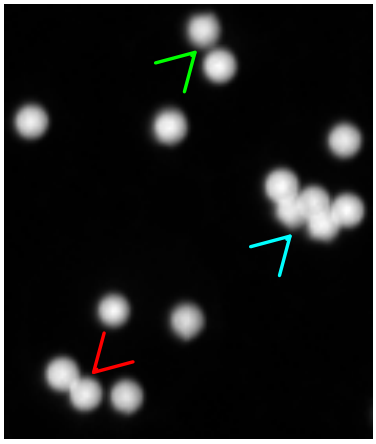

Figure 12: **Unwanted Microbead Phenomena**: The image is part of the `biopixR` package and illustrates the various occurrences within a microbead sample. It is evident that some microbeads are situated in proximity to one another, as indicated by the green arrow. Moreover, the microbeads have the potential to aggregate, forming clusters such as doublets (highlighted by the red arrow) and multiplets (marked by the cyan arrow).

The `proximityFilter()` function is used to filter objects based on a specified radius (Figure 13). For each object, a square region is defined around its center, representing an area in which no other object is permitted. In the event that another object is detected within this region, both objects are discarded. The user has the option to specify the radius in pixels, or alternatively, it can be calculated automatically by the algorithm. In the case of the latter, the algorithm assumes that the objects are circular and determines the radius based on their size. The calculated radius is then extended by a factor called `elongation`. The default elongation factor is 2, which means that an area of two radii (with one radius overlapping with the object) around the object must be free of other objects for the object to pass the filter. The `elongation` factor can be adjusted according to the user's specific requirements.

```
# Perform object detection on the 'beads' image using the 'edge' method
objects <-
  objectDetection(beads,
                  method = 'edge',
                  alpha = 1,
                  sigma = 0)

# Apply a proximity filter to the detected objects
filter_prox <-
```

```
  proximityFilter(objects$centers,
                  objects$coordinates,
                  radius = "auto",
                  elongation = 2)

# Change the pixel color of passing microbeads and highlight distinct objects in
# different colors
visual <- changePixelColor(
  beads,
  filter_prox$coordinates,
  color = factor(filter_prox$coordinates$value),
  visualize = F
)

plot(visual, axes = FALSE)
```

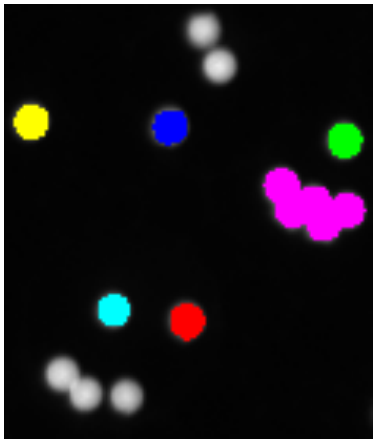

Figure 13: **Proximity Filtering Process**: The `objectDetection()` function is initially employed for the purpose of identifying all objects within the image. The resulting data from this function is then employed as input for the `proximityFilter()`, with the radius set to automatic calculation. The final result is presented through the use of the `changePixelColor()` function, wherein each passing object is highlighted in a distinct color.

The second phenomenon is the aggregation of microbeads, which results in the formation of doublets and multiplets. These aggregated microbeads must be discarded to achieve consistent and reproducible results

consisting of single microbeads. To address this issue, the `sizeFilter()` function was developed (Figure 14). This function filters objects based on their individual size, using specified lower and upper limits. These limits can be set manually, interactively, or automatically. In the interactive approach, the size distribution is plotted in the `R` environment, and the user is prompted in the console to enter the limits based on the visualized distribution. In the automated approach, which is applicable when the number of objects exceeds 50, the interquartile range (IQR) is used to calculate the limits. Specifically, the filter applies the 1.5 * IQR rule to determine the size thresholds. As described in several publications, the IQR rule is more suitable and reliable for outlier estimation when dealing with large data sets. Consequently, the conventional statistical threshold of n = 50 was selected (Miller 1991; Iglewicz and Hoaglin 1993; Seo 2006). This implies that at least 25 objects must be included in the IQR to produce a representative range.

```r
# Set up the plotting area to have 1 row and 2 columns
par(mfrow = c(1, 2))

# Apply a size filter to the detected objects from the previous chunk
filter_size <-
  sizeFilter(
    objects$centers,
    objects$coordinates,
    lowerlimit = 0,
    upperlimit = 150
  )

# Plot the sizes of the detected objects and display upper limit
plot(objects$centers$size, ylab = "size in px")
abline(h = 150, col = "red")           # Add a horizontal line at y = 150

# Change the pixel color of the passing microbeads and highlight distinct
# objects in different colors
visual <- changePixelColor(
  beads,
  filter_size$coordinates,
  color = factor(filter_size$coordinates$value),
  visualize = F
)

plot(visual, axes = FALSE)

# Reset the plotting area to the default 1 row and 1 column
par(mfrow = c(1, 1))
```

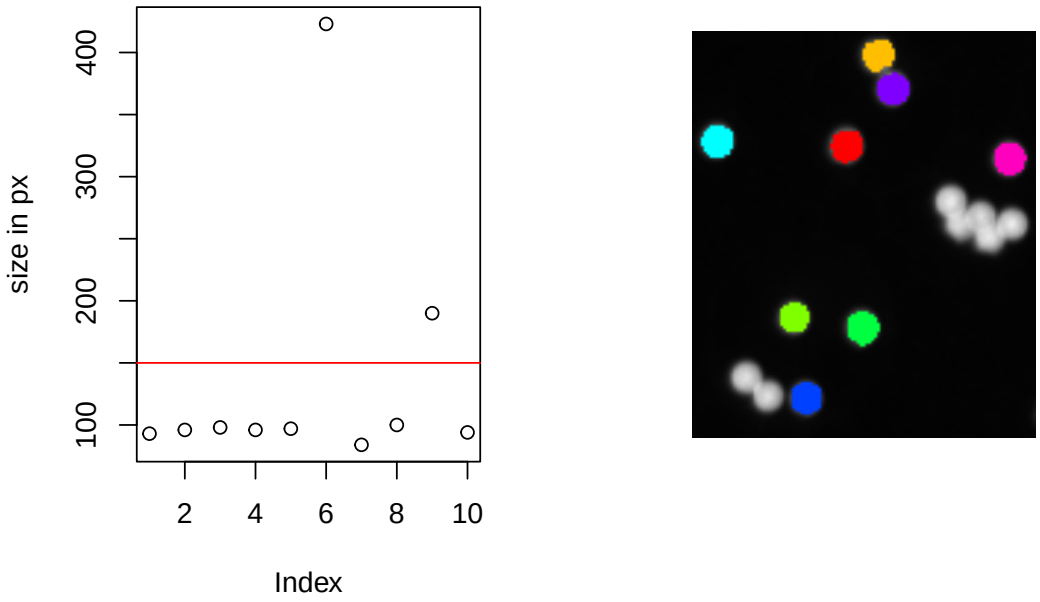

Figure 14: **Size Filtering Process**: The same input utilized for the previously demonstrated proximity filtering, obtained via the `objectDetection()` function, is employed in this instance. Due to an insufficient number of objects for automated calculation, the size limits are provided manually based on the size distribution shown on the left (in pixel - px). The provided limit is visualized through a red horizontal line. On the right, the `changePixelColor()` visualization tool is employed to highlight the objects that meet the size criteria in different colors.

As has been demonstrated, the `biopixR` package is an effective tool for addressing the undesired phenomena present in microbead-based assays. The package provides automated filter functions, some of which include interactive modules that facilitate applicability in laboratory settings.

4.5 Interpretation with `resultAnalytics()`

The `resultAnalytics()` function is designed to summarize the most important information about the extracted features. This includes details such as the number of objects, center coordinates, size, intensity, and the number of objects that did not pass the filtering process, if applied. It also provides coverage information, which indicates the percentage of the image that is considered part of the objects. The function takes the object coordinates and the image as input and calculates these properties.

The results are presented in the form of a list comprising two tables: one representing the average of all features, providing a comprehensive summary of the entire image in a single row (Table 1), and the second providing detailed information about each detected object (Table 2), which is useful but can be overwhelming when there are hundreds of objects in the image. Users can also provide the unfiltered coordinates if filtering was used; in this case, the function estimates the number of rejected objects based on the average size of all detected objects.

```
# Extract the results from the previous extraction and filtering in a concise manner
result <- resultAnalytics(beads,
                          coordinates = filter_size$coordinates,
                          unfiltered = objects$coordinates)
```

```
# Displaying the summarized results for the whole image
result$summary
```

Table 1: Summary of extracted features in an image.

number of objects	mean (size)	sd (size)	mean (intensity)	sd (intensity)	estimated rejected	coverage
8	94.8	4.86	0.59	0.18	6	0.047

Abbreviations: sd - standard deviation

```
# The output is obtained as list
# Displaying the detailed results for every microbead
result$detailed
```

Table 2: Detailed results of the individual extracted features.

objectnumber	size in px	intensity	sd(intensity)	x	y
1	93	0.577	0.177	63.97	8.68
2	96	0.630	0.189	69.01	20.24
3	98	0.593	0.183	9.23	37.97
4	96	0.628	0.183	53.27	39.62
5	97	0.591	0.177	108.66	43.86
7	84	0.606	0.177	35.39	97.61
8	100	0.531	0.167	58.69	101.17
10	94	0.567	0.169	39.50	125.00

Abbreviations: px - pixel, sd - standard deviation

4.6 Batch Proccesing Functions within the `biopixR` Package

This section presents two pipeline functions designed for the analysis of images. The initial function, termed `imgPipe()`, integrates a number of functions from the `biopixR` package, thereby enabling comprehensive analysis and filtering of an image through a single function. The second function, `scanDir()`, expands the functionality of `imgPipe()` to encompass batch processing. This functionality enables the analysis of entire directories, incorporating all the options, parameter adjustments, and individual filtering capabilities provided by the `imgPipe()` function.

4.6.1 `imgPipe()` - One Image, One Function

The `imgPipe()` function combines multiple functions into a unified pipeline, including `objectDetection()`, `sizeFilter()`, `proximityFilter()`, and `resultAnalytics()`. This function is capable to analyse a single image, but was also designed to be capable of processing multiple color channels concurrently. As an illustration, if an image contains objects that are distinguishable by color, with each object detectable in a separate channel, these images can be submitted for analysis using the `imgPipe()` function. The function will analyze both images and combine the results, providing a summary of the number of objects detected in each image. This feature is particularly advantageous for the analysis of dual-colored microbeads.

This section illustrates the application of the `imgPipe()` function. Figure 15 presents the initial state, which depicts the dual-colored microbead image (A) and the two single-color channels (B & C). The images are imported via the `importImage()` function, which automatically converts the imported TIFF image into a 'cimg' object. Subsequently, the images are transformed into grayscale and utilized as input for the `imgPipe()` function. Edge detection is employed for feature extraction, with the identical `alpha` and `sigma` parameters applied to both images (Figure 15B and C). It can be observed in Figure 15A that overlapping microbeads are present. To exclude these from the subsequent analysis, the `proximityFilter()` is enabled within the `imgPipe()` function. The output is in similar format as presented in Chapter 4.5, with the addition of a data frame comparing the differently encoded microbeads (Table 3 and 4).

```
# Import the dual-color microbead image
dual_color <- importImage("figures/fig19.2_dual_color.tif")

# Import the green fluorescence channel
green_beads <- importImage("figures/fig19.3_green_channel.tif")

# Import the red fluorescence channel
red_beads <- importImage("figures/fig19.1_red_channel.tif")
```

```
# Set up the plotting area to have 1 row and 3 columns
par(mfrow = c(1, 3))

# Plot the dual-color microbeads without axes
plot(dual_color, axes = FALSE)
text(c(60), c(60), c("A"), col = "darkred", cex = 5)

# Plot the green channel image without axes
plot(green_beads, axes = FALSE)
text(c(60), c(60), c("B"), col = "darkred", cex = 5)

# Plot the red channel image without axes
plot(red_beads, axes = FALSE)
text(c(60), c(60), c("C"), col = "darkred", cex = 5)
```

```r
# Reset the plotting area to the default 1 row and 1 column
par(mfrow = c(1, 1))
```

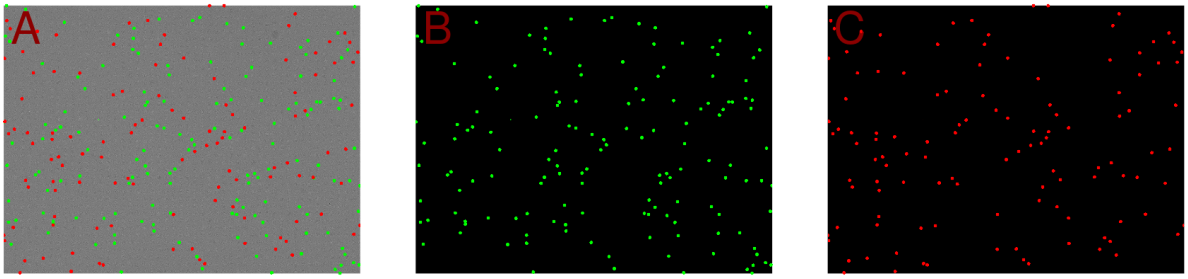

Figure 15: **Dual Colored Microbeads**: Images to be analyzed using the `imgPipe()` function. **A**) The image displays both microbead populations with different emission spectra, as indicated by the green and red signals. **B**) Shows the first input image with only the green emission detected. **C**) Shows the second input image with only the red emission detected. *Note*: *The microbeads in this image are only a few pixels in size and are highlighted in the respective color for easy identification.*

```r
# Discard second image present in image depth and transform into grayscale (green)
green_beads <-
  as.cimg(green_beads[1:dim(green_beads)[1],
                      1:dim(green_beads)[2],
                      1,
                      1:dim(green_beads)[4]]) |>
  grayscale()

# Discard second image present in image depth and transform into grayscale (red)
red_beads <-
  as.cimg(red_beads[1:dim(red_beads)[1],
                    1:dim(red_beads)[2],
                    1,
                    1:dim(red_beads)[4]]) |>
  grayscale()
```

```r
# Applying pipeline to analyse both fluorescence channels with enabled
# proximity filter
res_pipe <- imgPipe(green_beads,
                    color1 = "green",
                    red_beads,
                    color2 = "red",
                    method = 'edge',
                    alpha = 0.7,
```

```
                    sigma = 2,
                    sizeFilter = FALSE,
                    proximityFilter = TRUE)
```

Table 3: Summary of extracted features from dual-color microbeads

number of objects	mean (size)	sd (size)	mean (intensity)	sd (intensity)	estimated rejected	coverage	of color 1	of color 2
255	23.1	2.53	0.214	0.235	9	0.009	142	113

Abbreviations: sd - standard deviation

Table 4: Detailed information about encoded Objects

color	number of objects	mean(size)	sd(size)	mean(intensity)	sd(intensity)
green	142	23.3	2.10	0.278	0.025
red	113	22.9	2.98	0.130	0.018

Abbreviations: sd - standard deviation

To provide a conclusive visual representation of the findings, the original image, which depicts both populations of microbeads, is presented. Each detected microbead is indicated by a colored circle: microbeads with green emission are highlighted with a blue circle, and those with red emission are marked with an orange circle. Overlapping microbeads are not marked, as they are excluded from the analysis (Figure 16).

```
# Define a vector of colors
colors <- c("blue", "orange")

# Plot the dual-color microbead image
plot(dual_color, axes = FALSE)

# Add points to the plot at the center coordinates of the microbeads
# The points are colored based on their fluorescent signal (green/red)
points(res_pipe$detailed$x,
       res_pipe$detailed$y,
       col = colors[factor(res_pipe$detailed$color)],
       lwd = 3)
```

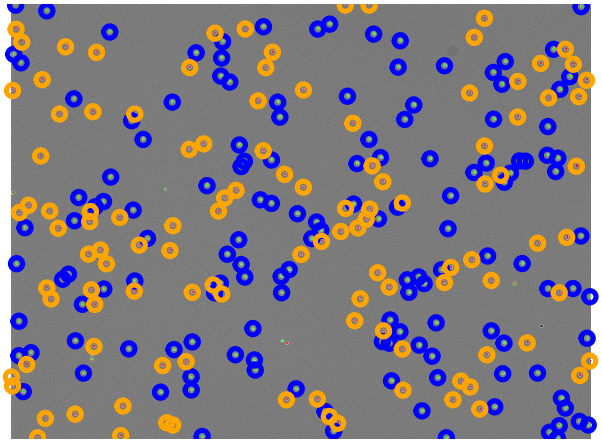

Figure 16: **Result of the Analysis of dual colored Microbeads**: The original image showing both microbead populations is presented. Detected microbeads with green emission are circled in blue, and those with red emission are circled in orange. Overlapping microbeads are not circled, due to their proximity and the resulting exclusion from the analysis.

In conclusion, this function provides a comprehensive pipeline for image analysis, encompassing both the initial preprocessing tasks, such as filtering, and the final analysis. The function may be applied to a single image devoid of encoded objects or to multiple images representing a single population of encoded objects.

4.6.2 `scanDir()` - Whole Directory Analysis

The `scanDir` function enables the analysis of images across entire directories. To ensure reliability and performance, the software includes a number of features designed to enhance the robustness and efficiency of the analysis process. These include double file checks via Message Digest 5 (MD5) sum comparison, multi-core processing capabilities, and the option to generate a log file that documents the analysis process and results.

MD5 sum - To increase the quality of the analysis, the function incorporates a verification process to check that the data is unique within the analysis directory. This is accomplished through the utilization of the MD5 algorithm, originally developed by Rivest (1992). The MD5 algorithm is a standard practice in computer science for ensuring file integrity and identifying identical files. The application of this algorithm results in the generation of a unique "fingerprint" for each file (Rivest 1992; Cechova 2020). In `R` this can be accomplished by invoking the `md5sum()` function from the `tools` package, which computes a 128-bit summary of the file contents, represented by 32 hexadecimal digits. Subsequently, the fingerprints are subjected to analysis to identify any duplicates. In the event that duplicate files are identified, the function

is terminated and the user is prompted to remove the duplicated files. However, due to its relatively short length (128-bit, 32 character, hexadecimal string) and the nature of hashing algorithms, it is possible for two different files to have the same MD5 value. This is called MD5 collision. In theory, this can be problematic in scenarios where hash values are used as a security measure or for data integrity checks since it may lead to false positives or negatives. However, we can additionally employ the file name and meta data (e.g., file creation date) alongside these hash values to counteract this issue.

Parallel Processing - In `R`, each image and its respective analysis script is sequentially processed on a single core of the Central Processing Unit (CPU), by default. This approach is especially time-consuming when analyzing multiple images from an entire experiment. The use of parallel processing in `R` allows for the simultaneous execution of multiple processes, thereby enhancing performance and speed. The use of packages such as `parallel`, `snow`, `foreach`, and `doParallel` allows computations to be distributed across multiple cores, within `R`. This is particularly advantageous for complex data analysis tasks and data-intensive applications, where parallel programming can significantly reduce computation time (Weston 2009; Schmidberger et al. 2009; Corporation and Weston 2011; R. D. Peng 2022).[4]

The `foreach` package provides a straightforward and efficient approach to implementing parallel processing. Consequently, this package was employed to enable parallel processing in the `scanDir()` function. The function was designed with the objective of achieving high parallelizability, thereby enabling each image to be analyzed on a separate core. For instance, if six cores are available, six images can be analyzed concurrently, thereby markedly accelerating the analysis process. The user may specify the number of cores to be utilized; alternatively, 75 % of all cores are used for computation.

The following section illustrates the functionality of the `scanDir()` function through an analysis of a directory containing the example images provided by the `biopixR` package (Figure 2). The 'threshold' method is employed without enabling any additional filtering processes. The resulting Table 5 will be saved in the working directory path in comma-separated value (CSV) format, if the `Rlog` parameter is set to `TRUE`.

```
# Get the path to the 'images' directory within the 'biopixR' package
path2dir <- system.file("images", package = "biopixR")

# Scan the directory for images and process them using the 'threshold' method
res_scanDir <- scanDir(
  path = path2dir,
  method = 'threshold',
  sizeFilter = FALSE,
  proximityFilter = FALSE,
  Rlog = FALSE
)
```

```
# Display an excerpt of the obtained results
# Showing columns 3 to 9 of the results, excluding file paths and md5 sums
res_scanDir[, 3:9]
```

[4] https://nceas.github.io/oss-lessons/parallel-computing-in-r/parallel-computing-in-r.html, accessed 06/27/2024

Table 5: Results obtained by the whole directory analysis.

	number of objects	mean (size)	sd (size)	mean (intensity)	sd (intensity)	estimated rejected	coverage
beads_large1	80	99.2	48.55	0.712	0.158	0	0.043
beads_large2	667	90.6	38.38	0.215	0.047	0	0.042
beads	9	127.7	86.19	0.678	0.142	0	0.071
droplet_beads	5	88.8	3.03	0.183	0.049	0	0.026

Abbreviations: sd - standard deviation

Log File - As one of the fundamental principles of this package is to facilitate reproducible research, the function documents the analysis process in an `RMarkdown` file and generates a comprehensive log file in Portable Document Format (PDF) format. `R` and `RMarkdown` are widely recognized tools for reproducible research (Baumer et al. 2014; Rödiger et al. 2015; Calero Valdez 2020). This package facilitates reproducibility through its open-source nature, which allows for the straightforward publication of analysis scripts. Furthermore, the log file provides visual quality control and comparability. Moreover, the detailed results (similar to Table 2) for each individual image within a directory are accessible via the `R` Data Serialization (RDS) files, saved as part of the logging process.

The following page (Figure 17) presents the log file generated from the `RMarkdown` file representing the results for this analysis. The directory of the analysis is first noted in the caption, followed by a series of logging steps that indicate the current status of the image being analyzed. Furthermore, the log file will indicate any instances of analysis failure resulting from errors in image processing. Subsequently, all images within the specified directory are plotted with the file name serving as the title. The detection of objects is indicated by green circles around their centers, providing an overview for verification of the analysis's compliance with expectations and the option of reanalyzing specific files if necessary.

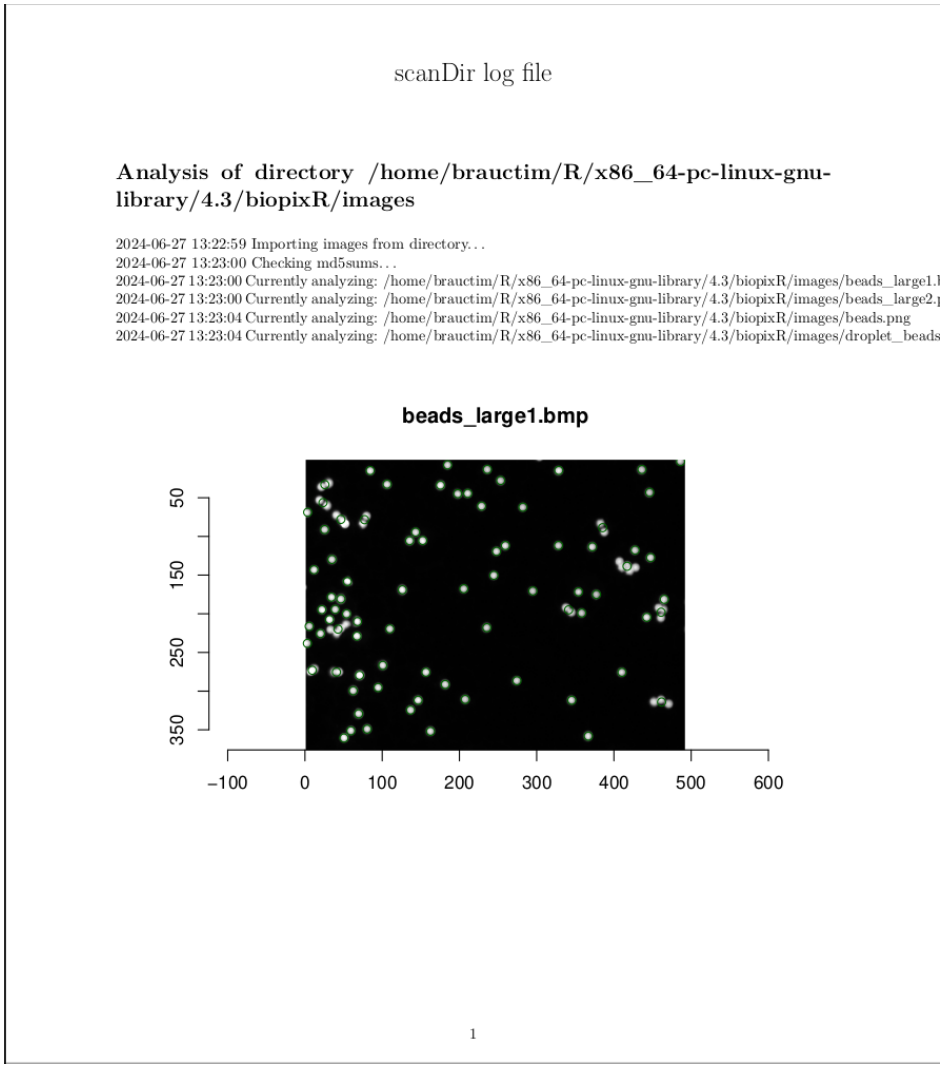

Figure 17: **Log file created by the `scanDir()` function.** (first page)

4.7 Functions for Droplet Analysis

In attempts involving microbead emulsions, the focus lies on situations where microbeads exist within a watery reaction environment that is surrounded by a hydrophobic matrix, particularly fluorinated oils. This results in the formation of small (sherical or non-spherical) microbead-water compartments with nanoliter to microliter volumes, which are often called partitions (Figures 18A). The objective of this approach is to separate individual microbeads from their surrounding reaction environments to avoid cross-reactions. Applications include the clonal amplification, single molecule detection and absolute quantification. Depending on the emulsification method employed, partitions of homogeneous size or heterogeneous (dispersed) size may be generated. (Rödiger et al. 2014).

Here, a common and challenging problem encountered in image analysis is the presence of discontinuous contours. This phenomenon presents a significant challenge to image processing tasks, such as segmentation, labeling, and feature extraction. Consequently, ensuring edge connectivity is an essential aspect of an effective

and reliable edge detector (Mittal et al. 2019). However, addressing low-level connectivity from the outset can present significant challenges, and there appear to be limited to no options available in `R` for addressing this issue. To address this issue, we propose the `fillLineGaps()` function.

4.7.1 `fillLineGaps()` - Restoring Edge-Connectivity in Compartmented Images

To demonstrate the functionality of the `fillLineGaps()` function in restoring edge connectivity, a practical example will be presented from a microbead-based emulsion Polymerase Chain Reaction (ePCR) assay. This area can benefit from imaging techniques, particularly to enhance applicability in Point-of-Care Testing (POCT), as the current method, Fluorescence-Activated Cell Sorting (FACS), is less suited for this purpose. The following example demonstrates the integration of data from two images to quantify droplets and microbeads. The objective is to identify the frequency of events where a single microbead joins a droplet in comparison to those where multiple microbeads are present within one droplet.

The images utilized in this section comprise a brightfield view and a fluorescent channel image. The brightfield view displays droplets with fragmented edges, some of which contain microbeads. In contrast, the fluorescent channel image exclusively reveals the microbeads (Figures 18A and B).

```
# Set up the plotting area to have 1 row and 2 columns
par(mfrow = c(1, 2))

# Plot the droplets containing microbeads without axes
plot(droplets, axes = FALSE)
text(c(10), c(10), c("A"), col = "darkred", cex = 6)

# Plot the fluorescence channel (only microbeads) without axes
plot(droplet_beads, axes = FALSE)
text(c(10), c(10), c("B"), col = "darkred", cex = 6)

# Reset the plotting area to the default 1 row and 1 column
par(mfrow = c(1, 1))
```

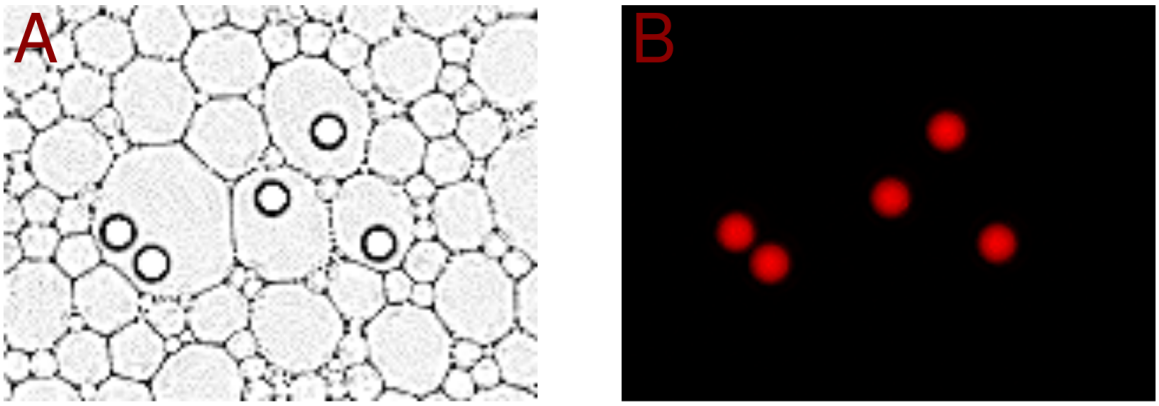

Figure 18: **Images from an ePCR Assay**: **A**) Brightfield view of a water-oil emulsion, showing droplets, some of which contain microbeads. The droplet contours are highly fragmented. **B**) Fluorescent channel of the same image, displaying only the microbeads.

The initial stage of the analysis involves the utilization of the `fillLineGaps()` function with the objective of restoring edge connectivity within the droplet image. This function identifies line endpoints and interpolates pixels to generate continuous contours. Initially, a threshold is applied to segment the contours and microbeads. Afterwards, object detection is performed on the fluorescence image to extract the coordinates of the objects, which are then excluded from the droplet image to prevent false reconnections between partition contours and microbeads.

Subsequently, an iterative scanning for line ends is initiated using the `image_morphology()` function from the `magick` package. The area surrounding the line ends is scanned within a specific radius, and reconnected with the closest contour using the `adaptiveInterpolation()` function. Following the reconnection phase, the contours undergo a thinning process, which is carried out by the `image_morphology()` function. These steps are repeated until the predefined number of iterations has been reached. The function includes an internal visualization that highlights the added pixels in purple (Figure 19). The method creates discrete partitions by closing the gaps in their contours, thus facilitating the correct labeling of the droplet partitions.

```
# Restoring edge connectivity
closed_gaps <- fillLineGaps(droplets,
  droplet_beads,
  threshold = "13%",
  alpha = 1,
  sigma = 0.1,
  radius = 5,
  iterations = 3,
  visualize = TRUE
)
```

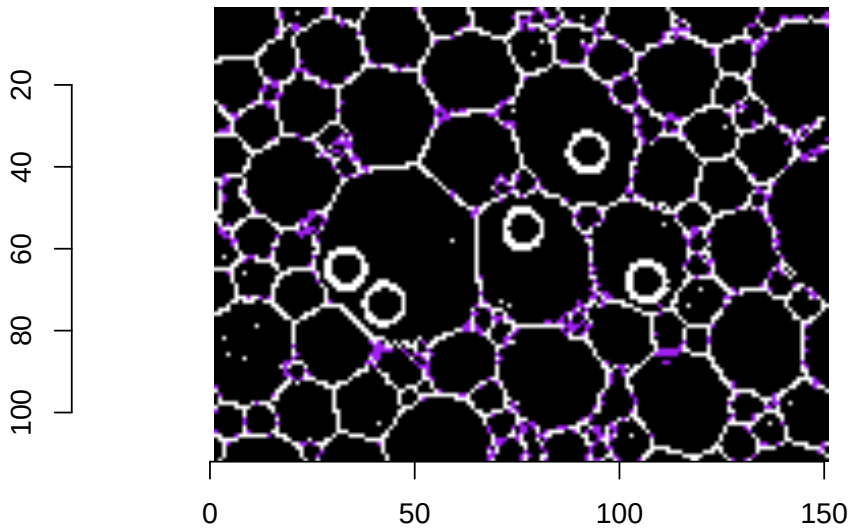

Figure 19: **Image of Droplets with Restored Edge Connectivity**: The result of utilizing the `fillLineGaps()` function is displayed. The pixels added by the function are highlighted in purple. It can be seen that the performance is accurate in simpler regions where straight lines are reconnected. However, in areas where smaller droplets are present between larger droplets, the reconstruction appears to be more challenging.

Having addressed the initial issue of discontinuous edges, the focus now shifts back to characterizing the partitions. This is achieved by labeling the 'closed_gaps' image. Once the partitions have been labeled and are therefore regarded as distinct, the next step is to ascertain the number of microbeads within each partition. This is achieved by utilizing the centroid coordinates of the microbeads, as opposed to analyzing each individual pixel. Each partition is then examined to identify which ones contain coordinates that correspond to the centers of the microbeads.

```
# Label the resulting image without discontinuous contours
lab_partitions <- label(closed_gaps)

# Convert the labeled droplet partitions to a data frame and keep values
# greater than 0
df_lab_part <- as.data.frame(lab_partitions) |>
  subset(value > 0)

# Perform object detection on the microbeads within the droplets
e_beads <- objectDetection(droplet_beads,
                           method = 'edge',
```

```
                          alpha = 1,
                          sigma = 2)

# Extract the relevant columns (containing coordinates)
coords1 <- df_lab_part[, 1:2]               # Coordinates of doplet partitions
coords2 <- round(e_beads$centers[, 2:3])    # Center coordinates of microbeads

# Convert the coordinates to character strings for easy matching
coords1_str <- apply(coords1, 1, paste, collapse = ",")
coords2_str <- apply(coords2, 1, paste, collapse = ",")

# Find the matching indices between the two sets of coordinates
matches <- which(coords1_str %in% coords2_str)

# Subset the data frame using the matching indices
bead_partition <- df_lab_part[matches, ]

# Create a table of the frequency of each partition value
numeration <- table(as.character(bead_partition$value))
```

The results are presented in Table 6. The first column of the Table specifies the number of partitions, and the second column lists those that are devoid of microbeads. Column three shows the number of partitions that contain microbeads. The final two columns indicate the frequency of occurrence of single microbeads within a droplet, as well as the occurrence of multiple microbeads within a single droplet. Figure 20 provides a visual representation of the result. The different microbead containing droplets are highlighted in distinct colors, demonstrating the successful restoration of partition contours and subsequent labeling process.

```
# Create resulting data frame highlighting the number of events with one
# microbead and multiple microbeads per droplet partition
res_df <- data.frame(
  partitions = length(unique(df_lab_part$value)),
  empty_partitions =
    length(unique(df_lab_part$value)) - length(unique(bead_partition$value)),
  bead_partitions = length(unique(bead_partition$value)),
  single_bead = length(which(numeration == 1)),
  muliple_beads = length(which(numeration > 1))
)
```

```
# Display resulting data frame
res_df
```

Table 6: Analysis of microbead-based ePCR

partitions	empty partitions	microbead partitions	single microbead	multiple microbeads
418	414	4	3	1

```
# Identify the droplets, which contain the microbeads
vis <- which(df_lab_part$value %in% bead_partition$value)

# Change the pixel color of the partitions containing microbeads and
# Highlight their distinctivness with different colors
colored_droplets <- changePixelColor(
  droplets,
  df_lab_part[vis, ],
  color = factor(df_lab_part$value[vis]),
  visualize = F
)

# Add the colored droplets and the marked microbeads to visualize the result
add(list(colored_droplets, e_beads$marked_objects)) |> plot(axes = FALSE)
```

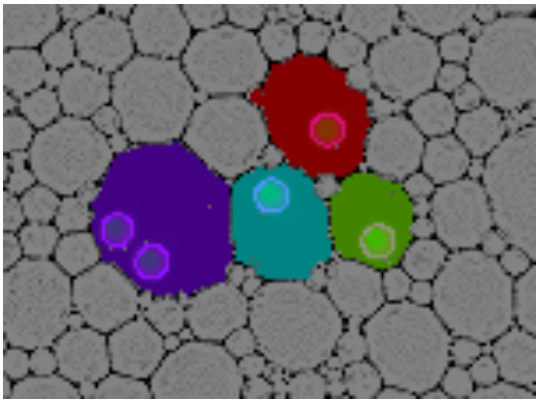

Figure 20: **Visualizing the Result of the Droplet Analysis**: The visualization obtained by the `objectDetection()` function, representing the detected microbeads, is combined with the highlighting of microbead-containing droplets using the `changePixelColor()` function. The partitions formed by the droplets are highlighted in different colors, indicating that they are distinct from one another.

In conclusion, this function displays considerable capacity to serve as a tool for image preprocessing, rendering more intricate images accessible for subsequent analysis. It enables the implementation of novel experimental approaches, as exemplified by the approach presented in this section. Moreover, it is currently the only available tool in `R` with a dedicated function for this purpose. It is important to note that the function does have certain limitations. One such limitation is that the restoration of connectivity can create small partitions that were not present in the original image.

4.8 Shape, Texture and unsupervised Machine Learning

Digital images are numerical representations of physical objects, combined with a point spread function (PSF). The PSF describes the intensity distribution of an image and serves as a physical reference, often utilized in image restoration, denoising, and object detection (Siddik et al. 2023; Song et al. 2024). This numerical representation enables a variety of computational approaches to gain insights into the object's characteristics, such as texture and shape extraction. To facilitate pattern recognition of these characteristics, the `biopixR` package incorporates two clustering algorithms that employ unsupervised machine learning techniques, namely Self-Organizing Maps (SOM) and the Partitioning Around Medoids (PAM) algorithm.

4.8.1 `shapeFeatures()` - Object Clustering Based on Shape Features

The `shapeFeatures()` function employs the `objectDetection()` function to extract objects from an image. Subsequently, these objects are analyzed with respect to their individual shape and intensity characteristics, including features such as pixel-intensity, area, perimeter, radius, eccentricity, circularity, and aspect ratio (AR). Subsequently, the extracted features are summarized by incorporating these characteristics into the `resultAnalytics()` output.

The `shapeFeatures()` function is capable of utilizing unsupervised machine learning techniques, specifically SOM. When SOMs are enabled, the function can classify detected objects based on their shape and intensity features, with the final output table including a row with classes that indicate the corresponding group for each object as determined by the SOM. This entails that the features are projected onto a two-dimensional plane, with similar features situated in proximity and dissimilar features positioned at a greater distance. The features are mapped to specific positions, designated as units, with each unit being associated with a codebook vector, representing the average of all features mapped to that unit. The number of these codebook vectors can be controlled through the `somegrid()` function of the `kohonen` package (Kohonen 1990, 2013; Wehrens and Kruisselbrink 2006, 2018). In `biopixR`, this is managed via the `xdim` and `ydim` parameters, providing the dimensions for the codebook vector gird. To utilize SOMs in `R`, the `kohonen` package is employed (Wehrens and Kruisselbrink 2006). The `biopixR` package, enables the usage of SOMs to access patterns in shape-related and pixel-intensity characteristics of image objects.

The aforementioned characteristics can be employed to distinguish image objects such as microbeads. The functionality of this classification is illustrated in Figure 21A, where non-circular objects (doublets and multiplets) are classified as a distinct group in comparison to single microbeads. The individual shape and intensity features of each microbead in the image are displayed as boxplots (Figure 21B). The characteristics of the microbeads marked with a red point in Figure 21A are also highlighted with a red point within the boxplot. The distinction between doublets and multiplets is readily apparent. Doublets and multiplets are

characterized by their larger size, perimeter, and radius, as well as lower circularity and higher eccentricity. This reflects their oval appearance, which differentiates them from single microbeads.

```
# Extract shape related features and group them using SOM
shape_features <-
  shapeFeatures(beads,
                alpha = 1,
                sigma = 0,
                xdim = 2,
                ydim = 1,
                SOM = TRUE)
```

```
par(mfrow = c(2, 1))

# Define a vector of colors
colors <- c("darkgreen", "darkred")

# Plot the example image containing microbeads
beads |> plot(axes = FALSE)

# Add solid circles to the plot with coordinates at the microbeads' centers
# The points are colored based on the 'class' factor in 'shape_features'
points(
  shape_features$x,
  shape_features$y,
  col = colors[factor(shape_features$class)],
  pch = 19,
  cex = 1.2
)
text(c(15), c(15), c("A"), col = "darkred", cex = 3.5)

# Create a data frame from 'shape_features' with selected columns
shape_df <- data.frame(
  size = shape_features$size,
  intensity = shape_features$intensity,
  perimeter = shape_features$perimeter,
  circularity = shape_features$circularity,
  eccentricity = shape_features$eccentricity,
  radius = shape_features$mean_radius,
  aspectRatio = shape_features$aspect_ratio
)

# Min-Max Normalization Function
min_max_norm <- function(x) {
```

```r
  (x - min(x)) / (max(x) - min(x))
}

# Apply the Min-Max Normalization function to each column of the data frame
df_normalized <- as.data.frame(lapply(shape_df, min_max_norm))

# Create a boxplot of the normalized data
boxplot(
  df_normalized,
  ylab = "normalized values",
  xaxt = "n"
)

# Add axis ticks and diagonal labels
axis(1, at = 1:ncol(shape_df), labels = FALSE)  # Add axis ticks but no labels
text(
  x = seq_len(ncol(df_normalized)),
  y = -0.1,
  labels = colnames(df_normalized),
  adj = 0,
  srt = -45,
  xpd = TRUE
)

# Extract rows to highlight where 'shape_features$class' equals 2
highlight_rows <-
  which(shape_features$class == 2)  # Example row indices to highlight

# Add points for the specific rows for each column
for (col in 1:ncol(df_normalized)) {
  points(
    rep(col, length(highlight_rows)),
    df_normalized[highlight_rows, col],
    col = "red",
    pch = 19,
    cex = 1.5
  )
}
text(c(0.5), c(0.9), c("B"), col = "darkred", cex = 3.5)
```

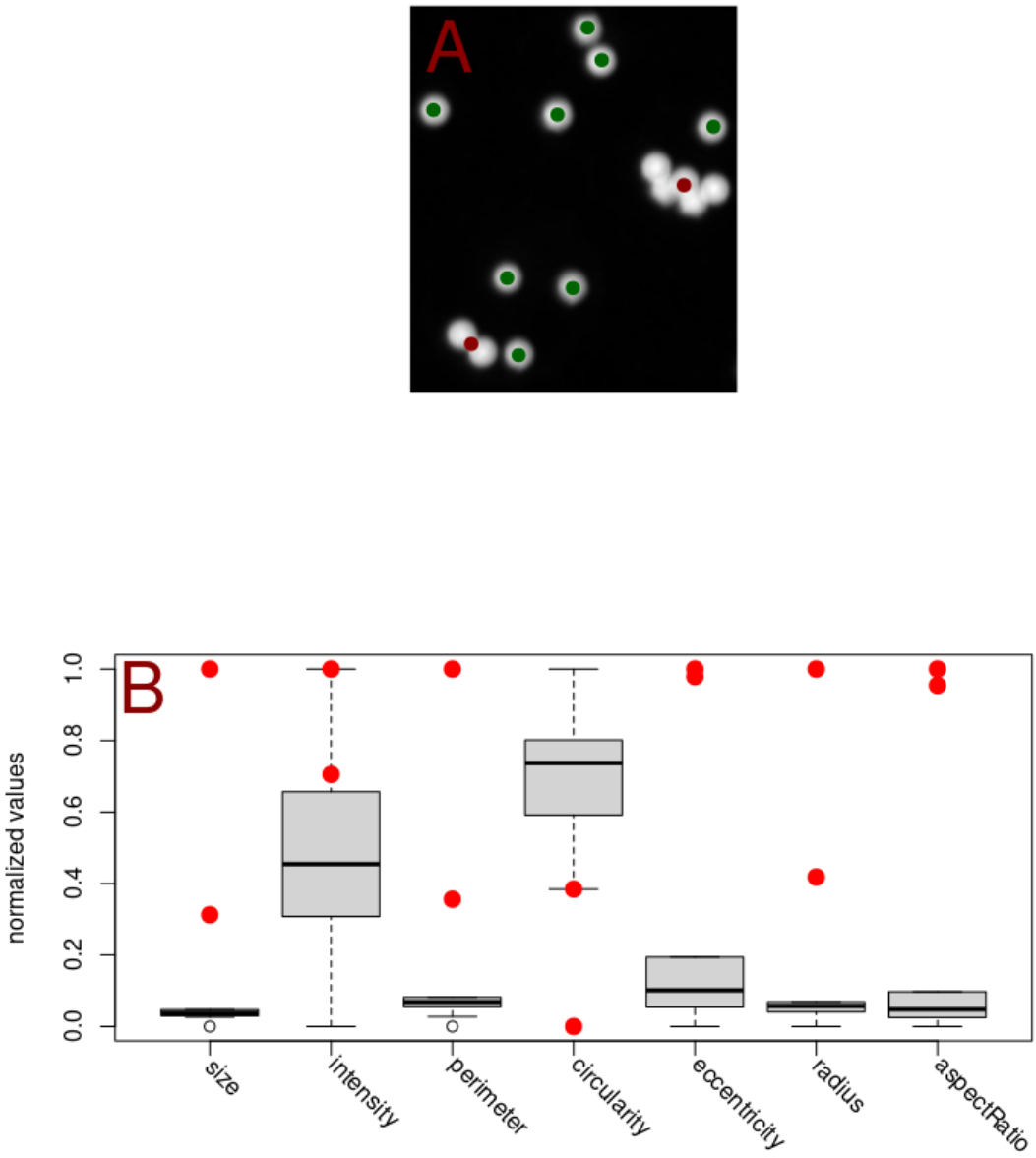

Figure 21: **Clustering of Microbeads Using SOM within `shapeFeatures()`**: **A**) This simple example demonstrates the use of the `shapeFeature()` function to group microbeads. One group is marked by a green point at its center, and the other group is marked by a red circle. **B**) The different features used as input characteristics for analysis by the SOM are plotted to compare the different groups. Features corresponding to objects marked by red circles in **A** are highlighted in red.

To validate the AR calculations performed using the `biopixR` package, a reference figure containing circles with known ground truth ARs was employed for comparison (Figure 22). The AR represents the ratio of an object's dimensions and can be described as follows (Takashimizu and Iiyoshi 2016):

$$AR = \frac{length\ of\ major\ axis}{length\ of\ minor\ axis} \tag{1}$$

A simplified method is utilized to calculate the major and minor axes. The distance from each perimeter pixel to the object's center is measured, with the largest distance representing the major radius and the smallest distance representing the minor radius. This function is based on the assumption that the object is symmetrical. The major and minor axes are then obtained by multiplying the respective radii by a factor of two. The validity of this simplification for symmetrical objects is illustrated in the following comparison.

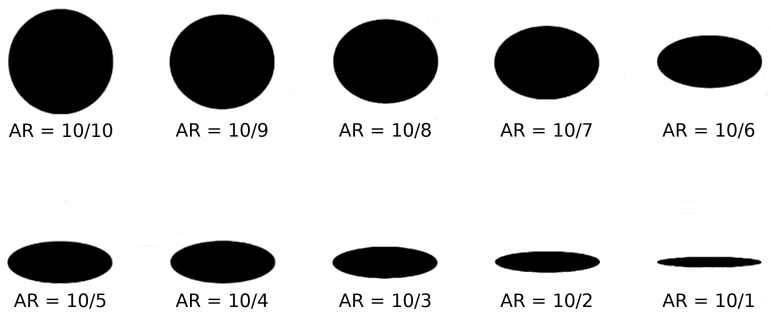

Figure 22: **Circles with different Aspect Ratios** (ARs): This is a modified image, originally derived from the publication of Takashimizu and Iiyoshi (2016). The enumeration of objects is read from left to right, beginning with the object in the top left (Table 7).

The results of the analysis of Figure 22 using the `shapeFeatures()` function are presented in Table 7, which demonstrates the ability to access shape-related information. It is noteworthy that circularity begins at a value of approximately one, which represents a perfect circle, and then decreases with an increasing AR. In contrast, eccentricity begins at zero and reaches 0.8 for the final circle. The radius represents the mean of the largest and smallest distances from a perimeter pixel to the microbeads' center. As the circle becomes more oval, the standard deviation increases, which can be explained by the aforementioned calculation. The AR is compared with the ground truth data (Figure 23). Notably, the calculations for the individual circles are accurate, with the exception of objects 5 and 6, where the AR deviates by 0.3 and 0.5, respectively.

```
# Import the image of different circles
circles <- importImage("figures/fig3_analysis_circ.png")

# Convert the imported image to grayscale
circles <- grayscale(circles)

# Extract shape related features from the grayscale image
shapes <- shapeFeatures(circles, alpha = 1, sigma = 0)
```

```r
# Display the obtained shape related information
data <- shapes[, c(1,8:12)]
```

Table 7: Comparison of aspect ratios.

objectnumber	circularity	eccentricity	mean(radius)	sd(radius)	aspect ratio
1	0.925	0.012	57.0	0.319	1.02
2	0.903	0.060	54.1	1.977	1.13
3	0.855	0.121	51.2	3.968	1.27
4	0.873	0.186	48.2	5.963	1.46
5	0.772	0.336	41.8	9.695	2.01
6	0.670	0.439	39.0	11.567	2.56
7	0.702	0.430	39.1	11.568	2.51
8	0.618	0.541	35.4	13.185	3.35
9	0.447	0.667	32.8	14.744	5.01
10	0.269	0.823	29.4	15.606	10.27

Abbreviations: sd - standard deviation

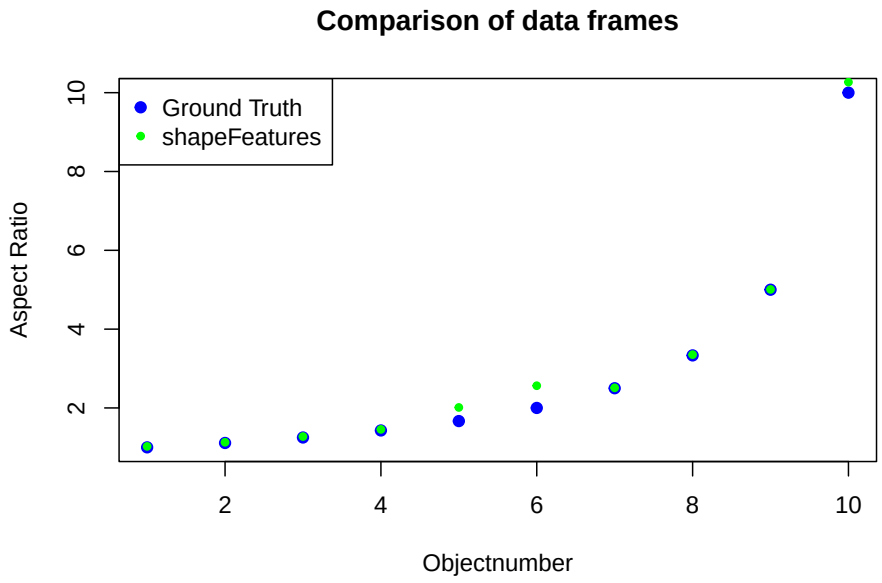

Figure 23: **Performance of the `shapeFeatures()` function in regards to the Aspect Ratio**: This plot compares the aspect ratio calculated by the `biopixR` package with ground truth data from a publication (Takashimizu and Iiyoshi 2016). The ground truth data is shown in blue, and the obtained results are shown in green.

In summary, the AR calculations performed by the `biopixR` package are found to be generally accurate

in comparison to the ground truth data, with most calculations demonstrating close alignment with the reference data. Minor discrepancies were observed at ARs of 10/6 and 2. The output obtained by the function is presented in Table 7. In the initial example, the `shapeFeatures()` function was successful in clustering the microbeads according to their shape characteristics, thereby demonstrating its utility as a tool for pattern recognition based on these features.

4.8.2 `haralickCluster()` - Image Classification Based on Texture Features

One aspect of interest in bioimage informatics is texture-related information. In bioimage informatics, texture refers to the spatial arrangement or pattern of structures within an image. It can provide important information like intensity variations, frequency content, and spatial relationships between pixels or regions in an image. Texture is crucial for various applications such as classification, segmentation, and diagnosis (e.g., to distinguish different microbeads, types of tissues, cells, or pathological conditions within histology images).

To extract this information from images, several computational methods were proposed by Haralick, Shanmugam, and Dinstein (1973). The `haralickCluster()` function incorporates several of these calculations, including contrast, angular second moment, correlation, variance, sum average, and entropy. As proposed by Haralick, Shanmugam, and Dinstein (1973), these texture features are employed for image classification in the `biopixR` package. Consequently, the `haralickCluster()` function incorporates a clustering algorithm, namely PAM.

The `haralickCluster()` function accepts a directory path as input, ensuring file uniqueness through the use of MD5 sums, as detailed in Chapter 4.6.2. To calculate the Haralick features, the image is first transformed into a Gray Level Co-occurrence Matrix (GLCM). Therefore an empty square matrix is generated with dimensions corresponding to the total number of gray levels in the image. This matrix records the frequency of specific pixel intensities being adjacent to each other. Subsequently, the matrix is normalized by dividing by the total number of co-occurrence pairs. Adjacency is considered in four directions: horizontal, vertical, left diagonal, and right diagonal (Haralick, Shanmugam, and Dinstein 1973; V 2012; Löfstedt et al. 2019). Subsequently, the features - contrast, angular second moment, correlation, variance, sum average, and entropy - are calculated. Some calculations were derived from the `radiomics` package, which also focuses on texture analysis but is currently not maintained.[5] Subsequently, the extracted features are clustered using the PAM algorithm.

The PAM algorithm consists of two parts: BUILD, which selects a specified number of objects as initial medoids, and SWAP, which improves the clustering process towards a local optimum by assigning each object to the closest medoid and recalculating the medoids to enhance within-cluster similarity. The algorithm starts this process by using a dissimilarity matrix as input which is then organized into defined clusters, with the medoids serving as robust representatives of cluster centers. The number of medoids is typically set with a predetermined number of groups (k) (Maechler et al. 1999; Reynolds et al. 2006; Schubert and Rousseeuw 2019). To utilize the PAM algorithm and to determine the number of k the `cluster` package was used (Maechler et al. 1999). The silhouette method, proposed by Rousseeuw (1987), provides a relative quality measure of the clustering, thereby providing a useful tool for approximating the optimal number of clusters (k). The `haralickCluster()` function in the `biopixR` package integrates the texture description of an image,

[5] https://CRAN.R-project.org/package=radiomics, accessed 06/30/2024

as proposed by Haralick, Shanmugam, and Dinstein (1973), with PAM clustering, enabling the classification of images in a directory based on these characteristics.

The following example illustrates the application of the `haralickCluster()` function to four images of microbeads, provided by the `biopixR` package (Figure 2). The resulting output is presented in Table 8. The results of the Haralick texture feature calculations for the different images, which served as input for the PAM algorithm, are presented in Figure 27. The results presented in Table 8 are represented in the plots, which illustrate the different texture features for the various images. Notably, *beads* and *beads_large1* are consistently quite similar, while *beads_large2* and *droplet_beads* also exhibit similarities with each other. However, these two groups differ between one another, as depicted in Figure 27.

```
# Get the path to the 'images' directory within the 'biopixR' package
path2dir <- system.file("images", package = "biopixR")

# Extract and group texture related characteristics based on Haralick texture features
img_clus <- haralickCluster(path2dir)
```

Table 8: Result of grouping according to texture features.

file name	md5sum	cluster
beads_large1.bmp	7b88193bf99c4efebb5e2e8f4b8066fe	1
beads_large2.png	b37015555151d6b387ea8a7ced30eaa3	2
beads.png	504d83fb571b8e8156f188b57733a60c	1
droplet_beads.png	67b542a4f0ec7f9f306dd9b0a4099dbc	2

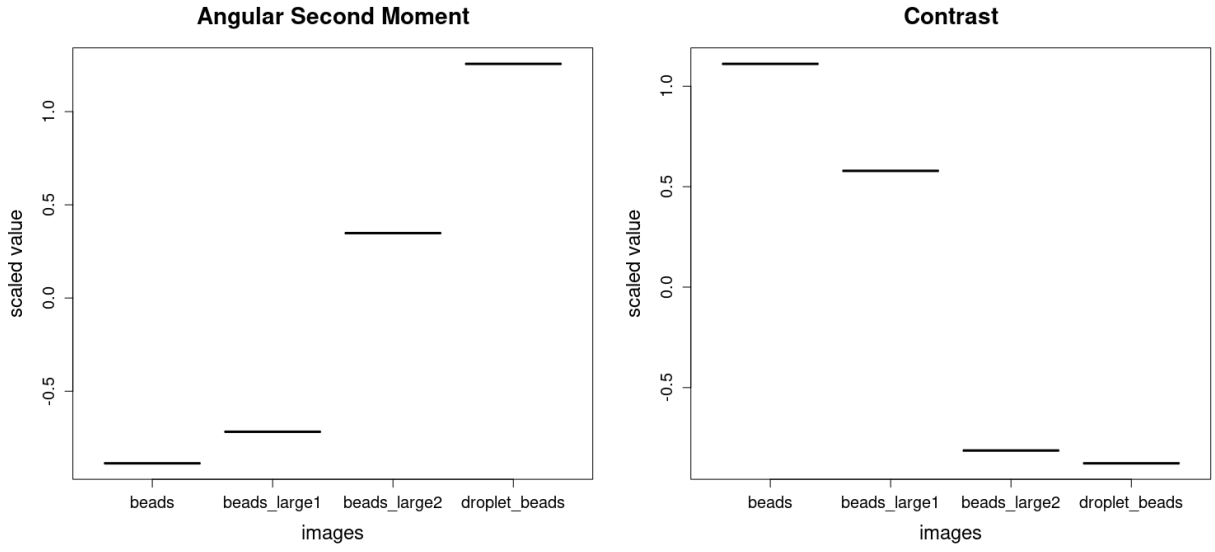

Figure 24: Part of Figure 27.

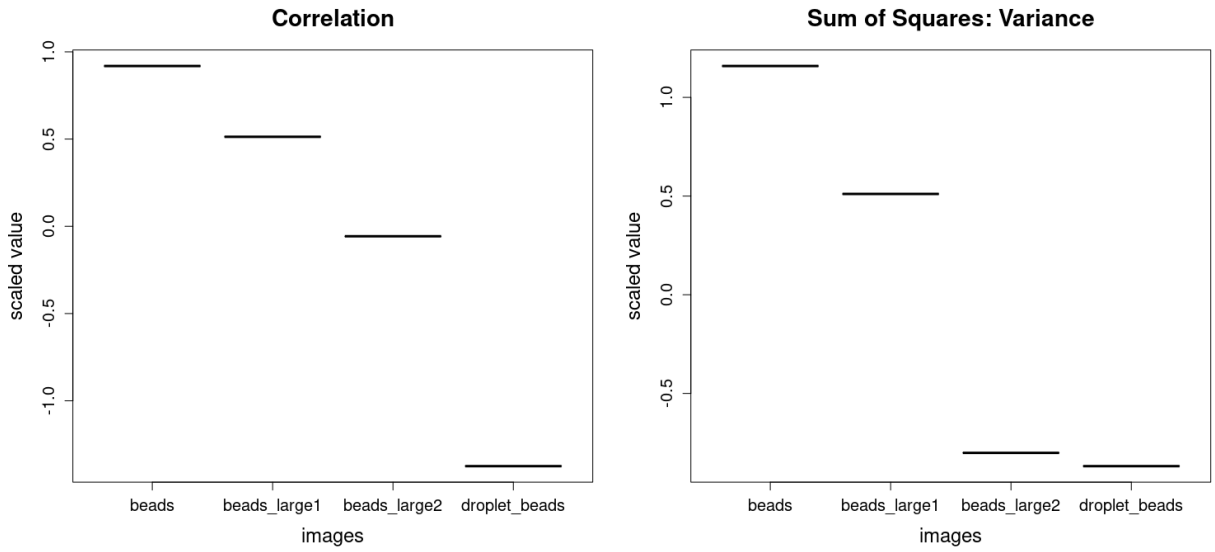

Figure 25: Part of Figure 27.

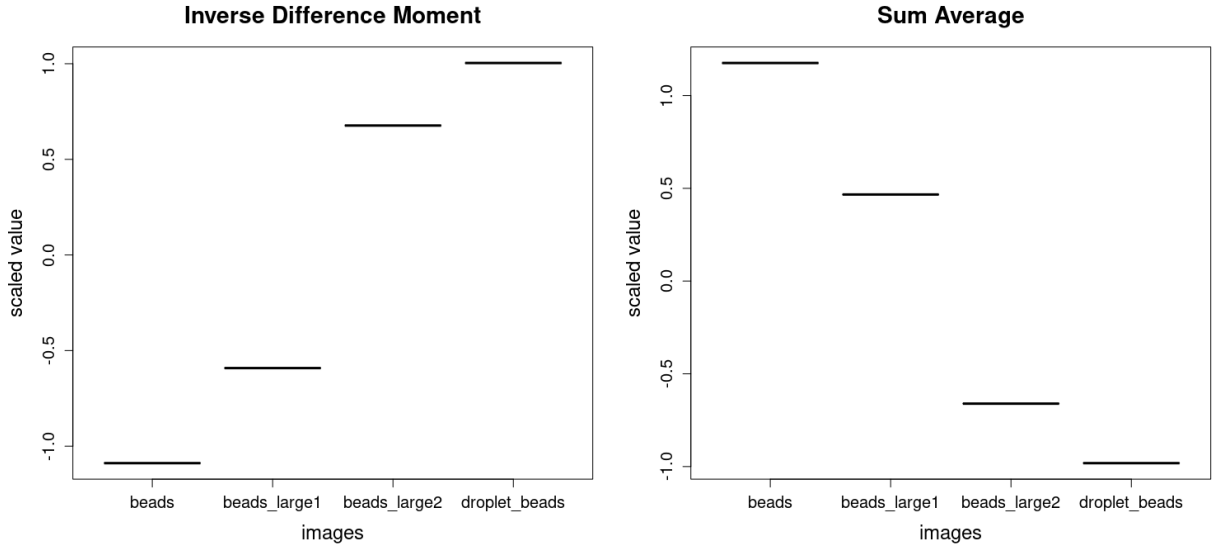

Figure 26: Part of Figure 27.

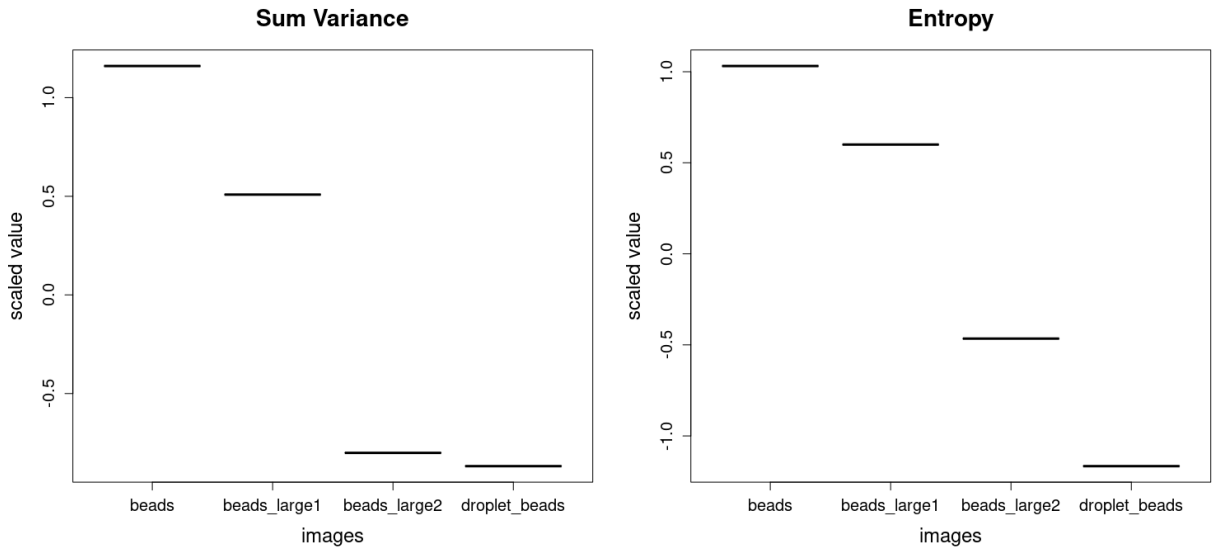

Figure 27: **Extracted Haralick Texture Features**: The scaled values of the calculated texture features for the four example images shown in Figure 2 are depicted. The results for the calculation of *Angular Second Moment*, *Correlation*, *Sum of Squares: Variance*, *Inverse Difference Moment*, *Sum Average*, *Sum Variance*, and *Entropy* are presented for each image. *Note*: *These values were obtained during the execution of the function and are not included in the function's output.*

To sum up, the `haralickCluster()` function is capable of extracting texture features and clustering images based on these features. To describe the texture characteristics of an image, the Haralick features were employed (Haralick, Shanmugam, and Dinstein 1973). For the purpose of clustering, a derivative of the *k-means* algorithm, PAM, was applied to classify the images according to the extracted features. The `haralickCluster()` function utilizes eight of the fourteen texture features proposed by Haralick, Shanmugam, and Dinstein (1973), thereby enabling the analysis and clustering of entire directories within a single, user-friendly function.

4.9 Helper Functions of the `biopixR` Package

The function `adaptiveInterpolation()` scans an increasing radius around the provided coordinates and connects them with the nearest labeled region. This function is designed to be incorporated into the `fillLineGaps()` function, which performs the thresholding and line end detection preprocessing. The line ends serve as coordinates and origins for interpolation. The `adaptiveInterpolation()` function generates a matrix with dimensions matching those of the original image. The initial matrix is populated solely with background values (0), corresponding to a black image. Subsequently, the function searches for line ends and identifies the nearest labeled region within a specified radius of the line end, excluding the cluster of the line end itself as a nearest neighbor. In the event that another cluster is identified, the `interpolatePixels()` function, which connects two points in a matrix, array, or image, is employed to connect the line end to this cluster. This results in the transformation of specific pixels within the matrix into foreground pixels with a value of 1.

The function `changePixelColor()` is a visualization tool that enables users to modify the color of a specified

set of pixels within an image. To perform this operation, the coordinates of the targeted pixels must be provided.

5 Discussion

The primary design objectives for the development of our package were to ensure low complexity, high maintainability, reliability, and a broad application spectrum, including automation features. Specifically, the objective was to reduce the number of dependencies on other packages in order to reduce overall complexity. Parallel processing and automation with Gaussian process models were incorporated to optimize the performance of pipelines for batch processing and medium-throughput analysis. In this chapter, we will assess the reliability of the `biopixR` package by comparing its performance with that of manual analysis data. Furthermore, we will evaluate its suitability for use in the field of cell biology, with a particular focus on the quantification of DNA damage.

5.1 Batch Processing and Big Data in `R`

Microbead technology represents a highly versatile tool for the implementation of highly parallelized quantitative multiparameter assays, which are capable of detecting nucleic acids and proteins. This technology offers several advantages, including low cost, minimal labor, high speed, and high-throughput automation, which make it particularly appealing for POCT applications (Rödiger et al. 2014; Zhang et al. 2019). Microbead assays are typically analyzed using fluorescence microscopy of a 96-well plate, with each well typically generating five images, resulting in up to 480 images for a single assay. Consequently, automation and batch processing are essential for analyzing data in this area of research.

Although `R` is a programming language that naturally supports the customized development of batch processing, we observed while collaborating with peers in wet laboratories that it is beneficial to offer certain routines as simple and easy-to-understand functions for average users. We have achieved this objective by implementing the functions `imgPipe()` and `scanDir()`. To the best of our knowledge, only the `FIELDimageR` and `pliman` packages provide comparable approaches, though they cater to very specific use cases distinct from ours (Matias, Caraza-Harter, and Endelman 2020; Olivoto 2022). Our approach enables users to analyze large data sets in medium-sized batches, depending on the computing performance. In an anecdotal example, we processed approximately 180 images with an average size of 1.4 MB (bmp, single channel) in 20 minutes using a computer with the specifications described in Chapter 3. This represents a substantial reduction in the time required for scientific work, as many scientists traditionally perform such tasks manually.

In the `R` programming language, the execution of for loops is a time-consuming, sequential process that iterates through each provided variable in a linear fashion. To enhance the efficiency of image data analysis, vectorization was employed. This approach permits the application of operations to entire vectors, which significantly accelerates the calculations. Additionally, the `foreach` package was utilized to further optimize the processing speed. This package enables parallel processing in `R`, allowing the utilization of multiple CPU cores, thereby markedly reducing the time required for computationally intensive tasks (Weston 2009; Corporation and Weston 2011). To enhance user-friendliness, the package supports the importation of

images in a variety of formats, which are automatically converted to the `imager` format, 'cimg'. The function facilitates the seamless importation of a variety of image formats (e.g., JPEG, PNG, BMP) from a single folder. This capability represents a distinctive feature of our batch processing function, allowing for the processing of images with different formats in a single batch. Moreover, the function checks for the presence of an alpha channel, which contains transparency information that can interfere with several functions. To mitigate potential issues, the import function ensures the removal of the alpha channel.

At this time, it is not evident that the existing implementation will yield any further significant performance improvements. The primary factors influencing the speed of processing are the speed of the CPU and the number of cores available. Given the absence of a suitable implementation for graphics chips, there is no potential for improvement in that area. However, the use of Hierarchical Data Format 5 (HDF5) could potentially result in enhanced speeds through the implementation of techniques such as chunking (including adjustments to the chunk cache size), compression to improve input/output (I/O) performance, parallelism at the CPU and file system levels, as well as access pattern optimization (Folk et al. 2011; Koranne 2011). This approach has not yet been subjected to evaluation and would be a task for future analysis projects.

Another objective for optimization could be the controlled import of images to prevent Random-Access Memory (RAM) overload. Given that `R` utilizes memory to store imported and generated data, directories containing a substantial volume of images can readily overload the RAM, leading to process termination and software crashes (Prajapati 2013). At present, the issue can only be mitigated manually by dividing the directory into multiple subdirectories and analyzing them individually. Future advancements should include strategies to avoid memory overload, such as reading images in batches. `R` provides a number of packages that address this problem, including `disk.frame` (ZJ and Poon 2019), `bigmemory` (Kane et al. 2008), and `ff` (Daniel Adler 2007). Nevertheless, the applicability of these packages for 'cimg-lists' has not been further evaluated and would also be considered a task for future projects.

5.2 Cyclomatic Complexity of `biopixR`

As previously stated, the `biopixR` package was designed with the intention of maintaining a minimalistic approach, with the objective of reducing dependencies and code complexity. It is of vital importance to ensure the reproducibility of software, not only for the advancement of our own work but also for the broader scientific community (Gentleman and Temple Lang 2007; Rödiger et al. 2015). One approach to achieving this objective involves minimizing dependencies on external packages or libraries, with the goal of relying on single archives whenever feasible. Consequently, the `biopixR` package requires the `R` programming language ($\geq$ 4.2.0), the `imager`, `magick`, and `tcltk` libraries, the `data.table` and `cluster` packages, and suggests the use of the `knitr`, `rmarkdown`, `doParallel`, `kohonen`, `imagerExtra`, `GPareto`, and `foreach` packages, all of which are available exclusively from CRAN.

A widely recognized metric used in both industry and research to evaluate code complexity is the cyclomatic complexity metric, also known as the McCabe Metric, which was introduced by McCabe (1976). This metric provides a quantitative measure of code complexity based on graph theory. Given the importance of ensuring long-term stability and reproducibility, it is crucial to maintain the complexity of individual package functions at a low level (McCabe 1976; Ebert et al. 2016).

The complexity of the `biopixR` package is quantified using the `cyclocomp` package, which measures the linearly independent paths through the code of a function, taking into account structures such as functions,

loops, and control statements (e.g., if, break, next, and return) (Csardi 2016).[6] Higher cyclomatic complexity values indicate more complex functions. The resulting complexity values for the functions of the `biopixR` package are presented in Table 9. The user-accessible functions are highlighted, while the helper functions, which are part of the main functions, are also listed. In the context of the `R` programming language, cyclomatic complexity can be regarded as an indicator of code maintainability, readability, and scalability. In general, lower values indicate code that is logically structured and easy to maintain, whereas higher values suggest more complex code that may be less maintainable.

```
# Load the 'kableExtra' library for enhanced table formatting
library(kableExtra)

# Calculate the cyclomatic complexity for the 'biopixR' package
cyclocomp_df <- cyclocomp::cyclocomp_package("biopixR")

# Remove row names from the data frame
rownames(cyclocomp_df) <- NULL
```

Table 9: Cyclomatic complexity in the 'biopixR' package. Cyclomatic complexity measures the number of linearly independent paths through the code, with higher scores indicating more complex code and lower scores indicating simpler, more maintainable code. The cyclomatic complexity was calculated using the cyclocomp package. Main functions accessible by the user are highlighted in bold, while other functions include helper functions and global variables that are part of the main functions.

name	cyclocomp
imgPipe	**42**
computeGLCM	32
haralickCluster	**32**
scanDir	**30**
objectDetection	**29**
proximityFilter	**21**
fillLineGaps	**18**
adaptiveInterpolation	**13**
interactive_objectDetection	**13**
sizeFilter	**12**
changePixelColor	**11**
edgeDetection	**9**
shapeFeatures	**7**
importImage	**6**

[6] https://github.com/MangoTheCat/cyclocomp, accessed 07/03/2024

rescueFill	5
resultAnalytics	**3**
wait_time_long	3
fillInit	1
guessKmeans	1
interpolatePixels	**1**
logIt	1
nonmax	1
print_with_timestamp	1
printWithTimestamp	1
value	1
x	1
y	1

In summary, all functions in the `biopixR` package have a cyclomatic complexity value of less than 50, with complexity increasing in the pipeline functions. In comparison, the `pliman` package, which offers a comparable pipeline for object quantification as described in Brauckhoff and Rödiger (to be published), has its `analyze_objects()` function scoring a cyclomatic complexity of 107.

5.3 Capabilities and Limitations - A Comparative Analysis of Human and Software Performance

A number of illustrative examples demonstrate that each individual microbead is accurately identified. However, the identification of aggregated microbeads, which are referred to as doublets or multiplets, does not align with the expected pattern. It is important to note that not every visually distinguishable microbead is regarded as a single object. The observed phenomenon, whereby doublets are identified as a single entity, is a consequence of the disappearance of their edges along the contact surface. The same principle applies to multiplets. Consequently, the software will undercount the number of objects in an image when aggregated microbeads are present. This variability could be addressed by integrating a watershed algorithm, a specialized segmentation algorithm capable of distinguishing touching objects (Beucher 1992; Pau et al. 2010). To demonstrate the discrepancy and comprehend the capabilities, an experiment was conducted to compare the manual analysis of three microbead images (Figure 28) (n=5) with the results obtained using the `biopixR` package.

A statistical comparison of the data will be conducted using the `R` package `irr`, which includes various methods for analyzing interrater reliability (Matthias Gamer 2005). To compare the manual and software-based methods, the Intraclass Correlation Coefficient (ICC) will be utilized. The ICC was originally introduced by Fisher (1992) and measures the correlation within a class of data, such as repeated measurements of the same objective. Typically, ICC values range between 0 and 1. Values below 0.5 indicate low reliability, while values above 0.8 or 0.9 indicate good to excellent reliability (Liljequist, Elfving, and Skavberg Roaldsen 2019). The ICC is based on the analysis of variances and, for the chosen one-way model, can be defined as

the ratio of variances (Bartko 1966). In the calculation, the software is considered to be one rater, and the mean of the five manual analyses is considered to be the other rater.

To further assess the differences between the two methods, a paired Student's t-test will be applied to determine whether there are any statistically significant differences in the obtained results. This test compares the mean differences and is designed for paired data, such as pre- and post-treatment measurements on a single individual (Hsu and Lachenbruch 2014). To perform the test, the single result obtained by the software was replicated five times, matching the five measurements derived from the manual analysis. Both methods are regarded as paired observations derived from the same images but employing different methods (treatments). To ensure the accuracy of this test, the data from the manual analysis will be tested for normal distribution using the Shapiro-Wilk normality test (Shapiro and Wilk 1965).

The manual analysis was conducted using a Shiny app, which is accessible at:

> https://brauckhoff.shinyapps.io/umfrage/; language: German

The survey was conducted by three colleagues in the same field of study (biotechnology), myself, and an independent individual from a completely different field of study. The initial objective of this survey was to enumerate each microbead, including those that are part of doublets or multiplets. The second task was to enumerate only the individual microbeads that are part of the aggregated structures. The aforementioned tasks were completed for three images, which are displayed in Figure 28. The images were selected for analysis because they depict varying amounts of microbeads and the phenomena of aggregated microbeads.

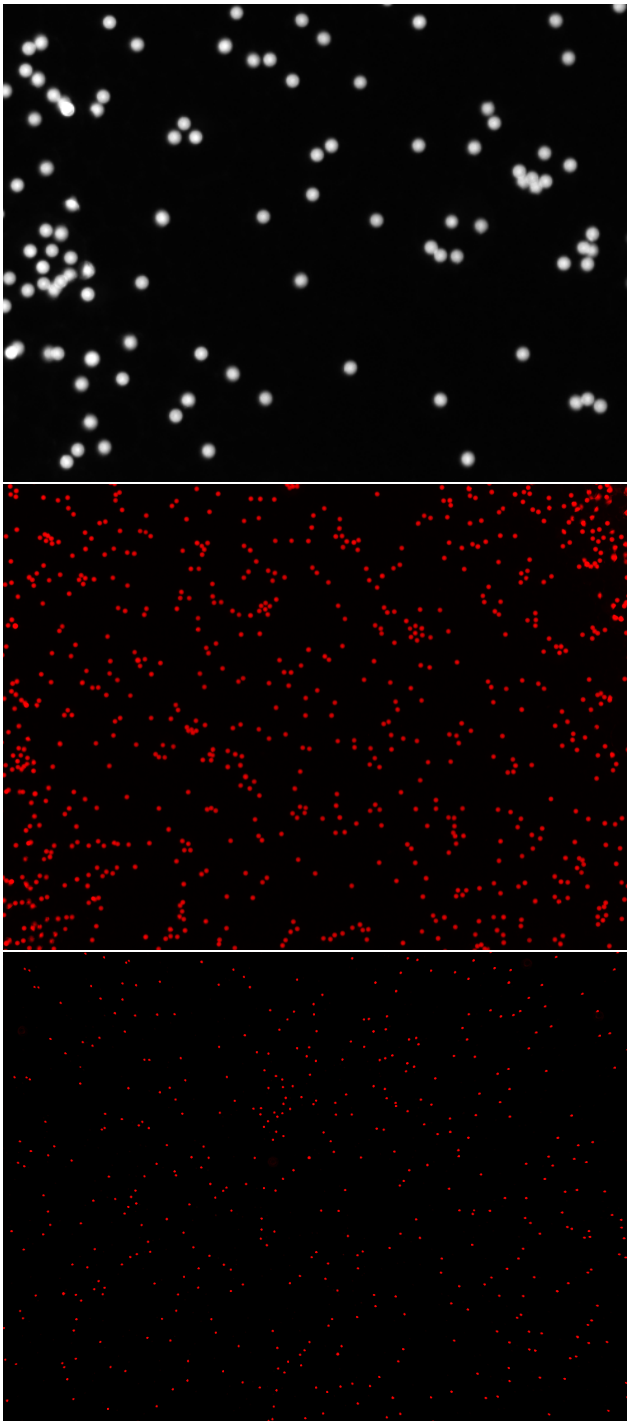

Figure 28: **Microbead Images for Manual Analysis**: These three images represent varying quantities of microbeads. Each image displays a unique combination of dimensions and the presence of doublets and multiplets at varying amounts. *Note: image1 - top; image2 - middle; image3 - buttom*

5.3.1 Preprocessing the Manual Analysis and Performing Analysis with `biopixR`

Initially, the data must be subjected to preprocessing in order to facilitate the display of the results. This process entails combining the results of the five individual analyses and running the `biopixR` package on the same images in order to obtain the comparative data frame. Subsequently, the two data frames, derived from the manual and software analysis, are merged for further processing. The individual processing steps are illustrated in the code below, and the final data utilized for analysis is depicted in Tables 10 and 11.

The design of the algorithm suggests that the software will yield lower count values for the first task. This discrepancy can be attributed to the algorithm's approach to doublets and multiplets, which are treated as single entities. In contrast, humans are able to visually distinguish individual microbeads even when they are part of doublets or multiplets. The results of task 1 are presented in Table 10. It is evident that manual counting for the image1 with fewer microbeads was accurate, with a standard deviation of 3.27. However, for the two larger images (Figure 28 image2 & 3) with higher numbers of microbeads, the standard deviation of the manual analysis increased to approximately 30. As anticipated, the algorithmic analysis produced a smaller object count than that obtained via manual counting for the first two images (Figure 28 image1 & 2). To interpret these findings, a statistical analysis was conducted and the results were visualized.

```
# Read CSV files containing manual count data
n1 <- read.csv("data/comparison/combined_data.csv")
n2 <- read.csv("data/comparison/combined_data_20240107084710.csv")
n3 <- read.csv("data/comparison/combined_data_20240302124005.csv")
n4 <- read.csv("data/comparison/combined_data_20240304210901.csv")
n5 <- read.csv("data/comparison/combined_data_20240701124906.csv")

# Calculate mean and standard deviation for manual analysis for each image (task 1)
manual_all <-
  data.frame(
    image1_all = c(
      mean(n1$Clicks[1], n2$Clicks[1], n3$Clicks[1], n4$Clicks[1], n5$Clicks[1]),
      sd(c(
        n1$Clicks[1], n2$Clicks[1], n3$Clicks[1], n4$Clicks[1], n5$Clicks[1]
      ))
    ),

    image2_all = c(
      mean(n1$Clicks[3], n2$Clicks[3], n3$Clicks[3], n4$Clicks[3], n5$Clicks[3]),
      sd(c(
        n1$Clicks[3], n2$Clicks[3], n3$Clicks[3], n4$Clicks[3], n5$Clicks[3]
      ))
    ),

    image3_all = c(
      mean(n1$Clicks[5], n2$Clicks[5], n3$Clicks[5], n4$Clicks[5], n5$Clicks[5]),
```

```
      sd(c(
        n1$Clicks[5], n2$Clicks[5], n3$Clicks[5], n4$Clicks[5], n5$Clicks[5]
      ))
    )
  )

# Calculate mean and standard deviation for manual analysis for each image (task 2)
manual_clott <-
  data.frame(
    image1_clot = c(
      mean(n1$Clicks[2], n2$Clicks[2], n3$Clicks[2], n4$Clicks[2], n5$Clicks[2]),
      sd(c(
        n1$Clicks[2], n2$Clicks[2], n3$Clicks[2], n4$Clicks[2], n5$Clicks[2]
      ))
    ),

    image2_clot = c(
      mean(n1$Clicks[4], n2$Clicks[4], n3$Clicks[4], n4$Clicks[4], n5$Clicks[4]),
      sd(c(
        n1$Clicks[4], n2$Clicks[4], n3$Clicks[4], n4$Clicks[4], n5$Clicks[4]
      ))
    ),

    image3_clot = c(
      mean(n1$Clicks[6], n2$Clicks[6], n3$Clicks[6], n4$Clicks[6], n5$Clicks[6]),
      sd(c(
        n1$Clicks[6], n2$Clicks[6], n3$Clicks[6], n4$Clicks[6], n5$Clicks[6]
      ))
    )
  )

# Transpose and modify column names
manual_all <- t(manual_all)
colnames(manual_all) <- c("Count", "sd")

manual_clott <- t(manual_clott)
colnames(manual_clott) <- c("Count", "sd")

# Import images for comparison with manual analysis
image1 <- importImage("figures/fig21.1_comparison.bmp")
image2 <- importImage("figures/fig21.2_comparison.png")
image3 <- importImage("figures/fig21.3_comparison.bmp")
```

```r
# Perform object detection on the images (comparison for task 1)
comparison1_img1 <- objectDetection(image1, alpha = 0.8, sigma = 0.7)
comparison1_img2 <- objectDetection(image2, alpha = 0.4, sigma = 0)
comparison1_img3 <- objectDetection(image3, alpha = 1, sigma = 0)

# Use pipeline function with size filtering to estimate amount of aggregated
# microbeads (comparison for task 2)
comparison2_img1 <- imgPipe(image1,
                            alpha = 0.8,
                            sigma = 0.7,
                            sizeFilter = T)
comparison2_img2 <- imgPipe(image2,
                            alpha = 0.4,
                            sigma = 0,
                            sizeFilter = T)
comparison2_img3 <- imgPipe(image3,
                            alpha = 1,
                            sigma = 0,
                            sizeFilter = T)

# Create a data frame for the first comparison
df_comparison1 <- data.frame(
  factor = c("image1", "image2", "image3"),
  software = c(
    length(comparison1_img1$centers$value),
    length(comparison1_img2$centers$value),
    length(comparison1_img3$centers$value)
  ),
  manual = as.numeric(manual_all[, 1]),
  manual_sd = as.numeric(manual_all[, 2])
)

# Create a data frame for the second comparison
df_comparison2 <- data.frame(
  factor = c("image1", "image2", "image3"),
  software = c(
    comparison2_img1$summary$estimated_rejected,
    comparison2_img2$summary$estimated_rejected,
    comparison2_img3$summary$estimated_rejected
  ),
  manual = as.numeric(manual_clott[, 1]),
  manual_sd = as.numeric(manual_clott[, 2])
)
```

Table 10: Comparison in object quantification for task 1. The table summarizes the quantification of all microbeads, including singlets and individual microbeads within doublets or multiplets. The analysis was conducted on the three example images presented at the beginning of this chapter. The 'biopixR' package was used for software-based quantification, while the manual analysis is presented as the average count (n = 5) with the standard deviation.

image	software	manual	sd(manual)
image1	87	100	3.27
image2	713	744	33.71
image3	439	424	28.06

Abbreviations: sd - standard deviation

Table 11: Comparison in object quantification for task 2. The table presents the results of counting individual microbeads within aggregated clusters, such as doublets or multiplets. The analysis was performed on the three aforementioned images using the 'biopixR' package and manual analysis (n = 5). For the manual analysis, the average count and standard deviation are reported.

image	software	manual	sd(manual)
image1	26	25	4.18
image2	144	89	28.89
image3	7	10	4.56

Abbreviations: sd - standard deviation

5.3.2 Visualization of Comparison between Human and Software Analysis

To assess the reliability of the software in comparison to manual analysis, the ICC was calculated. The count data for both tasks were subjected to a Shapiro-Wilk test to ascertain their normal distribution. The resulting p-values, all greater than 0.05, indicate that the data is normally distributed. Subsequently, a paired Student's t-test was conducted to compare the two methods. The results are presented in the corresponding plots for each task, with statistically significant differences marked with an asterisk (*) ($p < 0.05$).

In the first image, the software produced a significantly lower result compared to manual counting, which is an expected outcome. In contrast, for the other two images, the software results fall within the standard deviation of the manual analysis, with no statistically significant difference (Figure 29). The ICC calculated for the first task was 0.998, indicating excellent reliability between the results obtained by the software and those obtained through manual analysis.

```r
# Load the 'irr' library for calculating the ICC
library(irr)

# Calculate the ICC for the 'software' and 'manual' columns in df_comparison1
icc_result1 <- icc(df_comparison1[, 2:3])

# Print the ICC result
print(icc_result1)
```

```
 Single Score Intraclass Correlation

   Model: oneway
   Type : consistency

   Subjects = 3
     Raters = 2
     ICC(1) = 0.998

 F-Test, H0: r0 = 0 ; H1: r0 > 0
     F(2,3) = 894 , p = 6.86e-05

 95%-Confidence Interval for ICC Population Values:
  0.965 < ICC < 1
```

```r
# Vectorized approach for running Shapiro-Wilk test and checking normality
normality_p_values <- sapply(1:6, function(i) {
  shapiro.test(c(n1$Clicks[i],
                 n2$Clicks[i],
                 n3$Clicks[i],
                 n4$Clicks[i],
                 n5$Clicks[i]))$p.value
})

# Check normality and print results
for (i in 1:6) {
  if (normality_p_values[i] < 0.05) {
    message <- "Data is not normally distributed."
  } else {
    message <- "Data is normally distributed."
  }
  print(paste(
    "Test",
    i,
```

```
    ":",
    message,
    "(p-value =",
    round(normality_p_values[i], digits = 3),
    ")"
  ))
}
```

```
[1] "Test 1 : Data is normally distributed. (p-value = 0.544 )"
[1] "Test 2 : Data is normally distributed. (p-value = 0.257 )"
[1] "Test 3 : Data is normally distributed. (p-value = 0.377 )"
[1] "Test 4 : Data is normally distributed. (p-value = 0.834 )"
[1] "Test 5 : Data is normally distributed. (p-value = 0.82 )"
[1] "Test 6 : Data is normally distributed. (p-value = 0.544 )"
```

```
# Define indices for the paired t-tests
indices1 <- c(1, 3, 5)    # Rows represnting the results for task 1
indices2 <- 1:3           # Rows in comparative data frame for task 1

# Perform paired t-test for each selected pair of indices and collect p-values
p_values <- mapply(function(i, j) {
  t.test(
    rep(df_comparison1$software[j], 5),
    c(n1$Clicks[i], n2$Clicks[i], n3$Clicks[i], n4$Clicks[i], n5$Clicks[i]),
    paired = TRUE
  )$p.value
}, indices1, indices2)

# Significance threshold for the t-tests
significance_threshold <- 0.05

# Set up the plot area with custom x and y limits
plot(
  1:nrow(df_comparison1),
  df_comparison1$software,
  type = "n",
  xaxt = "n",
  xlim = c(0.5, nrow(df_comparison1) + 0.5),
  ylim = range(
    c(
      df_comparison1$software,
      df_comparison1$manual + df_comparison1$manual_sd
    )
```

```
  ),
  main = "Comparison of Methods",
  xlab = "Factors",
  ylab = "Counts"
)

# Add custom x-axis labels
axis(1,
     at = 1:nrow(df_comparison1),
     labels = df_comparison1$factor)

# Add points for the 'software' method
points(
  1:nrow(df_comparison1),
  df_comparison1$software,
  pch = 16,
  col = "darkcyan",
  cex = 1.5
)

# Add points for the 'manual' method
points(
  1:nrow(df_comparison1),
  df_comparison1$manual,
  pch = 16,
  col = "orange",
  cex = 1.5
)

# Add error bars for the 'manual' method
arrows(
  x0 = 1:nrow(df_comparison1),
  y0 = df_comparison1$manual - df_comparison1$manual_sd,
  x1 = 1:nrow(df_comparison1),
  y1 = df_comparison1$manual + df_comparison1$manual_sd,
  angle = 90,
  code = 3,
  length = 0.1,
  col = "orange"
)

# Add asterisks to indicate significant differences based on p-values
for (i in 1:nrow(df_comparison1)) {
```

```
  if (p_values[i] < significance_threshold) {
    text(
      i,
      max(df_comparison1$software[i], df_comparison1$manual[i]) + 40,
      "*",
      cex = 1.5,
      col = "black"
    )
  }
}

# Add legend to the plot
legend(
  "topright",
  legend = c("Software", "Manual"),
  col = c("darkcyan", "orange"),
  pch = 16
)
```

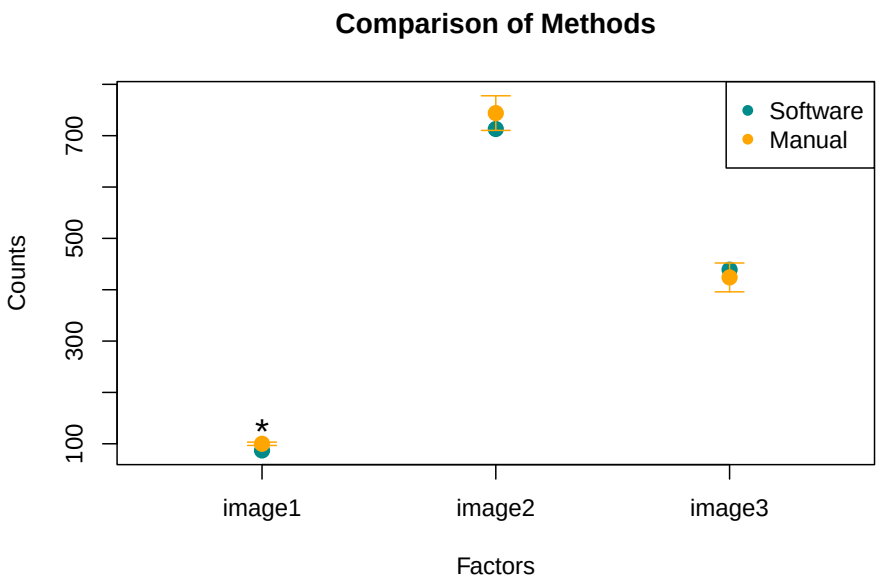

Figure 29: **Comparison of Software and Manual Object Quantification** (Task 1): All microbeads, including those in doublets or multiplets, are quantified manually by five different individuals (n = 5) and by the software `biopixR`. The resulting counts for each method are shown, with software results in darkcyan and manual analysis results in orange. The manual analysis is shown as mean ± standard deviation. *Note*: *Statistical significance was calculated using a paired Student's t-test, with * $p < 0.05$.*

The second task is to estimate the number of microbeads present within doublets and multiplets. The data for this analysis is presented in Table 11. The results from both the software and manual analysis of images

one and three are comparable, with a low standard deviation for the latter. The first image, with lower image dimensions, and the third, with a low number of doublets and multiplets, are comparatively straightforward to analyze. In contrast, the second image depicts a larger number of these structures, resulting in greater variability in the results obtained by manual analysis, with a standard deviation of nearly 30 (Figure 28 and Table 11). In order to create a comparable software result for this task, the `imgPipe()` function with an enabled `sizeFilter()` was employed. The algorithm estimates the number of rejected microbeads within these structures by dividing the number of discarded pixels by the mean size of the remaining objects (microbeads). The aforementioned results are illustrated in Figure 30, which demonstrates that the software and manual analysis yield nearly identical outcomes for images one and three. However, the differing outcome for image two is illustrated, with a statistically significant difference between the methods. The calculated ICC for task 2 indicates a high degree of reliability between the methods, with a score of 0.86.

```r
# Same for task 2
# Load the 'irr' library for calculating the ICC
library(irr)

# Calculate the ICC for the 'software' and 'manual' columns in df_comparison2
icc_result2 <- icc(df_comparison2[, 2:3])

# Print the ICC result
print(icc_result2)
```

```
 Single Score Intraclass Correlation

   Model: oneway
   Type : consistency

   Subjects = 3
     Raters = 2
     ICC(1) = 0.86

 F-Test, H0: r0 = 0 ; H1: r0 > 0
     F(2,3) = 13.3 , p = 0.0322

 95%-Confidence Interval for ICC Population Values:
  -0.092 < ICC < 0.996
```

```r
# Check for normality see analysis of task 1

# Perform paired t-test for each selected pair of indices and collect p-values
p_values <- sapply(1:nrow(df_comparison2), function(i) {
  t.test(
    rep(df_comparison2$software[i], 5),
    c(n1$Clicks[i * 2],
```

```
      n2$Clicks[i * 2],
      n3$Clicks[i * 2],
      n4$Clicks[i * 2],
      n5$Clicks[i * 2]),
    paired = TRUE
  )$p.value
})

# Significance threshold for the t-tests
significance_threshold <- 0.05

# Set up the plot area with custom x and y limits
plot(
  1:nrow(df_comparison2),
  df_comparison2$software,
  type = "n",
  xaxt = "n",
  xlim = c(0.5, nrow(df_comparison2) + 0.5),
  ylim = range(
    c(
      df_comparison2$software,
      df_comparison2$manual + df_comparison2$manual_sd
    )
  ),
  main = "Comparison of Methods",
  xlab = "Factors",
  ylab = "Counts"
)

# Add custom x-axis labels
axis(1,
     at = 1:nrow(df_comparison2),
     labels = df_comparison2$factor)

# Add points for the 'software' method
points(
  1:nrow(df_comparison2),
  df_comparison2$software,
  pch = 16,
  col = "darkcyan",
  cex = 1.5
)
```

```r
# Add points for the 'manual' method
points(
  1:nrow(df_comparison2),
  df_comparison2$manual,
  pch = 16,
  col = "orange",
  cex = 1.5
)

# Add error bars for the 'manual' method
arrows(
  x0 = 1:nrow(df_comparison2),
  y0 = df_comparison2$manual - df_comparison2$manual_sd,
  x1 = 1:nrow(df_comparison2),
  y1 = df_comparison2$manual + df_comparison2$manual_sd,
  angle = 90,
  code = 3,
  length = 0.1,
  col = "orange"
)

# Add asterisks to indicate significant differences based on p-values
for (i in 1:nrow(df_comparison2)) {
  if (p_values[i] < significance_threshold) {
    text(
      i,
      max(df_comparison2$software[i], df_comparison2$manual[i]) - 90,
      "*",
      cex = 1.5,
      col = "black"
    )
  }
}

# Add legend to the plot
legend(
  "topright",
  legend = c("Software", "Manual"),
  col = c("darkcyan", "orange"),
  pch = 16
)
```

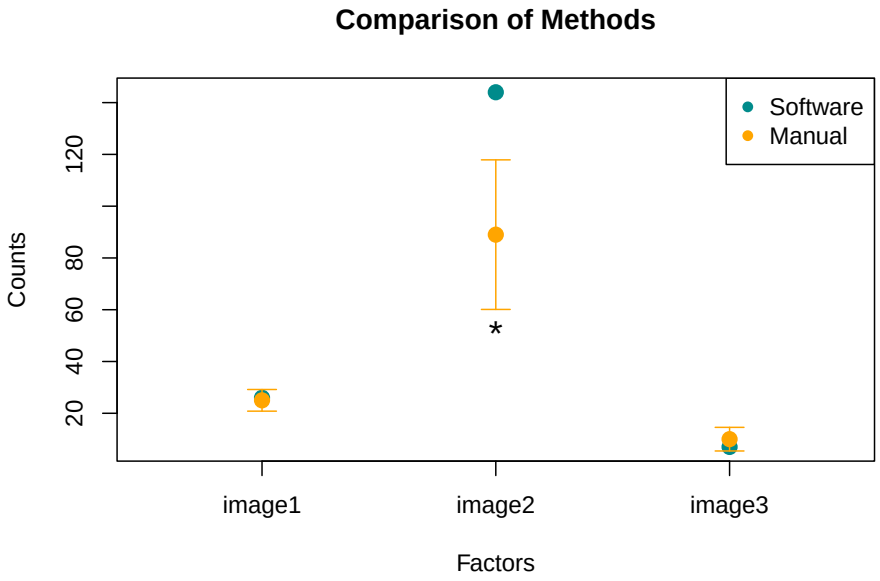

Figure 30: **Comparison of Software and Manual Object Quantification** (Task 2): Only microbeads that are part of doublets or multiplets are counted. Manual analysis was conducted by five different individuals (n = 5) and compared to the software `biopixR`. The resulting counts for each method are shown, with software results in darkcyan and manual analysis results in orange. The manual analysis is depicted as mean ± standard deviation. *Note*: *Statistical significance was calculated using a paired Student's t-test, with* * $p < 0.05$.

In conclusion, as the complexity of images increases, as evidenced by an elevated number of objects and the occurrence of phenomena such as doublets and multiplets, manual analysis becomes increasingly inaccurate, resulting in high standard deviations and compromised reproducibility. This highlights the necessity of the `biopixR` package for the reproducible quantification of image objects. In terms of reliability, the results obtained using the `biopixR` package are comparable to those obtained through manual analysis, but without the variation introduced by human error, as indicated by the calculated ICC. In conclusion, this section demonstrates the significance of the `biopixR` package in enhancing the reliability, quality, and reproducibility of object quantification in image analysis.

5.4 Exploring New Areas of Applicability

The `biopixR` package was initially designed and developed with a primary focus on microbeads. However, its applicability extends beyond this area of research. The package can be utilized for any image analysis task requiring feature extraction of spherical or round objects, including fields such as microplastic analysis and tracking (Bannerman and Wan 2016; Ding et al. 2020) and cell biology (Schneider et al. 2019). To illustrate the diverse applications of the `biopixR` package beyond microbeads, we present an example of its use in analyzing images obtained from cell biology studies.

DNA Double-strand breaks (DSBs) represent a particularly severe form of DNA damage, frequently resulting in apoptotic cell death in the absence of repair. The quantification of these breaks can be achieved through

immunofluorescence staining, which employs antibodies directed against the phosphorylated histone protein H2AX (γH2AX). This staining technique results in the formation of γH2AX foci, which serve as a quantitative representation of the number of DNA DSBs. The number of DSBs is proposed to reflect the efficacy of anti-tumor agents and, therefore, facilitate the assessment of individual patient responses to therapies. Furthermore, the number of DSBs may be used to evaluate the general cytotoxic effects of treatments *in vivo*. This approach allows for the precise modulation of therapy according to the individual needs of patients (Rödiger et al. 2018; Ruhe et al. 2019; Schneider et al. 2019).

In the following example, the `biopixR` package was employed to highlight extracted γH2AX foci. To quantify the green fluorescent γH2AX foci, the green color channel was initially extracted to enable the distinct identification of each individual foci using the `objectDetection()` function. The extracted foci were then visualized using the `changePixelColor()` function, with a unique color assigned to each distinct foci (Figure 31B). Subsequently, the number of foci per cell was determined in order to assess the state of the culture or tissue. This entailed the segmentation of cells through the application of the `objectDetection()` function to the blue color channel, which represents the DAPI-stained nuclei. By comparing the coordinates, the number of foci per cell was extracted. As illustrated in Figure 32, the majority of cells exhibit no DNA damage, with 27 % displaying a single foci and a small proportion demonstrating higher levels of damage with multiple foci.

```
# Import the image from the specified file path
DSB_img <- importImage("figures/tim_242602_c_s6c1+2+3m3.tif")

# Extract the green channel from the image representing yH2AX
yH2AX <- DSB_img[, , , 2] |> as.cimg()

# Perform object detection on the green channel image using the 'edge' method
# with specified parameters alpha and sigma
DSB <-
  objectDetection(yH2AX,
                  method = 'edge',
                  alpha = 1.3,
                  sigma = 0)

# Change the pixel colors of the detected objects
colored_DSB <-
  changePixelColor(yH2AX,
                   DSB$coordinates,
                   color = factor(DSB$coordinates$value),
                   vis = FALSE)

# Plot the imported image without axes
plot(DSB_img, axes = FALSE)

# Add a text annotation "A" at coordinates (50, 60) with dark red color and size 6
```

```
text(c(50), c(60), c("A"), col = "darkred", cex = 6)

# Plot the image with colored objects without axes
plot(colored_DSB, axes = FALSE)

# Add a text annotation "B" at coordinates (50, 60) with dark red color and size 6
text(c(50), c(60), c("B"), col = "darkred", cex = 6)
```

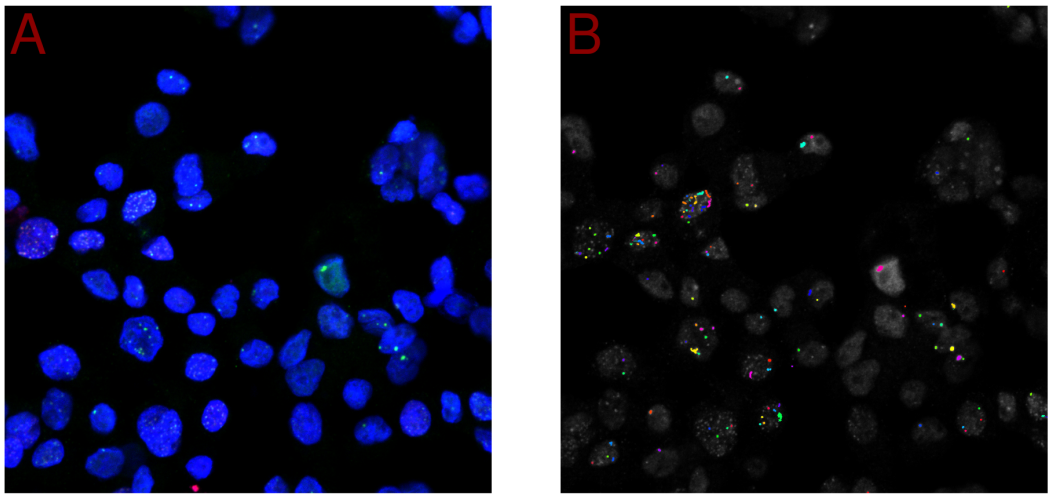

Figure 31: **Visualization of the Application of `biopixR` in Cell Biology**: **A**) The image shows HepG2 cells with nuclei stained using DAPI. The quantitative marker for DNA double-strand breaks, γH2AX, is targeted with a specific antibody and appears as green fluorescent foci. The experimental procedure adheres to the method described in Rödiger et al. (2018). **B**) The γH2AX foci are quantified using the `biopixR` package, with the detected foci highlighted in different colors by the `changePixelColor()` function.

```
# Extract the blue channel from the image representing nuclei
core <- DSB_img[, , , 3] |> as.cimg()

# Perform object detection on the blue channel using the 'threshold' method
cores <- objectDetection(core, method = 'threshold')

# Function to compare coordinates from two data frames
compareCoordinates <- function(df1, df2) {
  # Create a single identifier for each coordinate pair in both data frames
  df1$coord_id <- paste(round(df1$mx), round(df1$my), sep = ",")
  df2$coord_id <- paste(df2$x, df2$y, sep = ",")

  # Find matches between the coordinate identifiers
```

```
  matches <- df2$coord_id %in% df1$coord_id

  DT <- data.table(df2)
  DT$DSB <- matches

  # Summarize the results
  result <-
    DT[, list(count = length(which(DSB == TRUE))), by = value]

  return(result)
}

# Compare coordinates between detected DSB centers and cores' coordinates
count <- compareCoordinates(DSB$centers, cores$coordinates)

# Extract the counts for further analysis
to_analyze <- count[, 2]

# Create a frequency table of the counts
event_counts <- table(to_analyze)

# Create a barplot of the frequency distribution of yH2AX foci per cell
barplot(
  event_counts,
  xlab = "foci per cell",
  ylab = "frequency",
  cex.axis = 1.2,
  cex.lab = 1.4,
  cex.main = 1.6
)
```

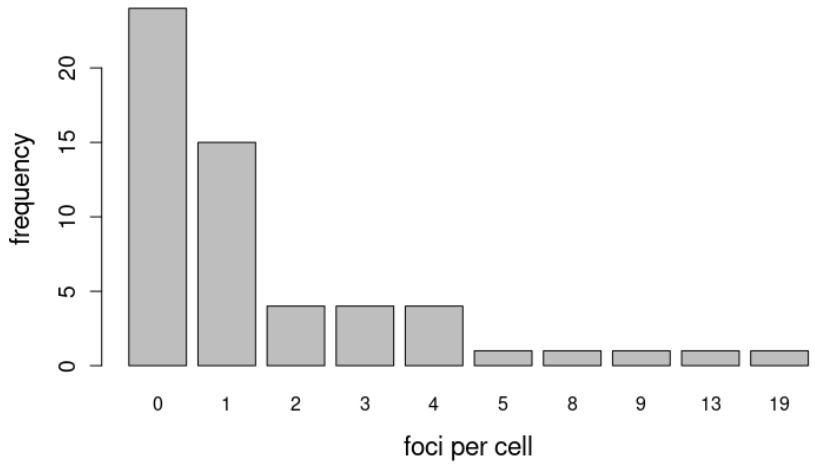

Figure 32: **Quantification of γH2AX Foci per Cell**: The bar plot displays the results of quantitative foci extraction from cellular images using the `biopixR` package. It shows the distribution of γH2AX foci per cell, revealing that most cells exhibit few or no DNA double-strand breaks, indicated by the absence of γH2AX foci. The frequency of events (number of foci per cell) is plotted to illustrate this distribution.

In conclusion, this illustrative example of DNA damage assessment illustrates the capabilities of the `biopixR` package in cell biology. The utilization of edge detection for foci extraction in conjunction with thresholding for nuclei identification serves to demonstrate the suitability of `biopixR` for complex imaging tasks. The integration of multiple segmentation strategies within a unified function enhances the flexibility of `biopixR`, rendering it applicable across diverse fields within the life sciences.

6 Summary and Conclusion

The present extended vignette demonstrates the efficacy and versatility of the `biopixR` package for a multitude of tasks involving the analysis of image data, including microbead-based and cell-based assays. Specifically, we have demonstrated that software-based object detection is capable of identifying objects with greater speed and efficiency, while quantifying features from images in a more reliable manner compared to manual analysis. Given that the `biopixR` package is work in progress, there is scope for optimization and further improvements. Ongoing testing continues to reveal bugs, which will be addressed in a timely manner. Some of the issues identified during the writing process include a problem with the `scanDir()` function, where the log function is not able to locate the documentation file. Another area for improvement is the optimization of the `haralickCluster()` function by replacing loops with vectorized operations, as the current use of loops results in slower performance. Moreover, the enhancement of unit tests and the integration of the quality control package `covr` into the continuous integration workflow will also contribute to an improvement in code quality. Long-term objectives include the development of an interactive Shiny application to provide a graphical web interface for the functions available in `biopixR`, thereby enhancing the program's accessibility to a broader user base. Another objective is to enhance the package's applicability for analyzing double-strand breaks, building on initial experiments that have already demonstrated its potential in this area.

7 Acknowledgement

We would like to express our gratitude to Dr. Franziska Dinter (BTU Cottbus - Senftenberg, Senftenberg, Germany) and Dr. Coline Kieffer (PSL University, Paris, France) for providing the microbead images, conducting the related experiments and discussion. Their contributions were essential for the development and testing of the `biopixR` package. Additionally, our appreciation goes to those who performed the manual analysis for comparing the developed software against human performance: Julius Rublack, Juliane Hohlfeld, Niclas Anschütz, and Tim Stübner.

```
sessionInfo()
```

```
R version 4.3.2 (2023-10-31)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 22.04.3 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.10.0
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=de_DE.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=de_DE.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=de_DE.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=de_DE.UTF-8 LC_IDENTIFICATION=C

time zone: Europe/Berlin
tzcode source: system (glibc)

attached base packages:
[1] tcltk     stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
[1] irr_0.84.1       lpSolve_5.6.20   kableExtra_1.4.0 biopixR_1.1.0
[5] magick_2.8.3     imager_1.0.2     magrittr_2.0.3   foodwebr_0.1.1
[9] knitr_1.46

loaded via a namespace (and not attached):
 [1] tidyselect_1.2.1   viridisLite_0.4.2  dplyr_1.1.4        fastmap_1.1.1
 [5] pracma_2.4.4       digest_0.6.35      lifecycle_1.0.4    cluster_2.1.6
 [9] processx_3.8.4     KrigInv_1.4.2      compiler_4.3.2     rlang_1.1.4
[13] tools_4.3.2        igraph_2.0.3       utf8_1.2.4         yaml_2.3.8
[17] data.table_1.15.4  htmlwidgets_1.6.4  mclust_6.0.1       xml2_1.3.6
[21] DiceDesign_1.10    KernSmooth_2.23-22 withr_3.0.0        purrr_1.0.2
[25] desc_1.4.3         grid_4.3.2         rgenoud_5.9-0.10   rgl_1.2.8
[29] fansi_1.0.6        colorspace_2.1-0   emoa_0.5-2         scales_1.3.0
[33] MASS_7.3-60        readbitmap_0.1.5   cli_3.6.2          mvtnorm_1.2-4
[37] crayon_1.5.2       rmarkdown_2.26     remotes_2.4.2.1    generics_0.1.3
[41] rstudioapi_0.16.0  stringr_1.5.1      parallel_4.3.2     tiff_0.1-12
[45] base64enc_0.1-3    vctrs_0.6.5        Matrix_1.6-4       jsonlite_1.8.8
```

```
[49] bookdown_0.37      callr_3.7.6        fftwtools_0.9-11   kohonen_3.0.12
[53] systemfonts_1.0.5  jpeg_0.1-10        DiceKriging_1.6.0  cyclocomp_1.1.1
[57] tidyr_1.3.0        glue_1.7.0         ps_1.7.6           rngWELL_0.10-9
[61] stringi_1.8.4      randtoolbox_2.0.4  munsell_0.5.1      imagerExtra_1.3.2
[65] tibble_3.2.1       pillar_1.9.0       anMC_0.2.5         htmltools_0.5.8.1
[69] R6_2.5.1           pso_1.0.4          GPareto_1.1.8      ks_1.14.1
[73] tidygraph_1.3.1    bmp_0.3            evaluate_0.24.0    pbivnorm_0.6.0
[77] lattice_0.22-5     png_0.1-8          tictoc_1.2         Rcpp_1.0.12
[81] svglite_2.1.3      xfun_0.43          pkgconfig_2.0.3
```

References

Appleton-Fox, Lewin. 2022. "Foodwebr: Visualise Function Dependencies." https://github.com/lewinfox/foodwebr.

Bååth, Rasmus. 2012. "The State of Naming Conventions in R." *The R Journal* 4 (2): 74–75. https://doi.org/10.32614/RJ-2012-018.

Bannerman, Dawn, and Wankei Wan. 2016. "Multifunctional Microbeads for Drug Delivery in TACE." *Expert Opinion on Drug Delivery* 13 (9): 1289–1300. https://doi.org/10.1080/17425247.2016.1192122.

Barthelme, Simon. 2015. "Imager: Image Processing Library Based on 'CImg'." *CRAN: Contributed Packages*, August. https://doi.org/10.32614/cran.package.imager.

Barthelmé, Simon, and David Tschumperlé. 2019. "Imager: An R Package for Image Processing Based on CImg." *Journal of Open Source Software* 4 (38): 1012. https://doi.org/10.21105/joss.01012.

Bartko, John J. 1966. "The Intraclass Correlation Coefficient as a Measure of Reliability." *Psychological Reports* 19 (1): 3–11. https://doi.org/10.2466/pr0.1966.19.1.3.

Baumer, Ben, Mine Cetinkaya-Rundel, Andrew Bray, Linda Loi, and Nicholas J. Horton. 2014. "R Markdown: Integrating a Reproducible Analysis Tool into Introductory Statistics." *Technology Innovations in Statistics Education* 8 (1). https://doi.org/10.5070/t581020118.

Beare, Richard, Bradley Lowekamp, and Ziv Yaniv. 2018. "Image Segmentation, Registration and Characterization in R with SimpleITK." *Journal of Statistical Software* 86 (8). https://doi.org/10.18637/jss.v086.i08.

Beucher, Serge. 1992. "The Watershed Transformation Applied to Image Segmentation." *Scanning Microscopy* 1992 (6): 28.

Binois, Mickaël, and Victor Picheny. 2019. "GPareto: An R Package for Gaussian-Process-Based Multi-Objective Optimization and Analysis." *Journal of Statistical Software* 89 (8). https://doi.org/10.18637/jss.v089.i08.

Blischak, John D., Emily R. Davenport, and Greg Wilson. 2016. "A Quick Introduction to Version Control with Git and GitHub." Edited by Francis Ouellette. *PLOS Computational Biology* 12 (1): e1004668. https://doi.org/10.1371/journal.pcbi.1004668.

Brauckhoff, Tim, and Stefan Rödiger. to be published. "Exploring Image Analysis with R: Applications and Advancements," to be published.

Caicedo, Juan C, Sam Cooper, Florian Heigwer, Scott Warchal, Peng Qiu, Csaba Molnar, Aliaksei S Vasilevich, et al. 2017. "Data-Analysis Strategies for Image-Based Cell Profiling." *Nature Methods* 14 (9): 849–63. https://doi.org/10.1038/nmeth.4397.

Calero Valdez, André. 2020. "Making Reproducible Research Simple Using RMarkdown and the OSF." In *Lecture Notes in Computer Science*, 27–44. Springer International Publishing. https://doi.org/10.1007/978-3-030-49570-1_3.

Canny, John. 1986. "A Computational Approach to Edge Detection." *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8 (6): 679–98. https://doi.org/10.1109/tpami.1986.4767851.

Cao, Jingwen, Ran Xu, Fuhan Wang, Yuan Geng, Tianchao Xu, Mengran Zhu, Hongli Lv, Shiwen Xu, and Meng-yao Guo. 2023. "Polyethylene Microplastics Trigger Cell Apoptosis and Inflammation via Inducing Oxidative Stress and Activation of the NLRP3 Inflammasome in Carp Gills." *Fish &Amp; Shellfish Immunology* 132 (January): 108470. https://doi.org/10.1016/j.fsi.2022.108470.

Cechova, Monika. 2020. "Ten Simple Rules for Biologists Initiating a Collaboration with Computer Scientists." Edited by Scott Markel. *PLOS Computational Biology* 16 (10): e1008281. https://doi.org/10.

1371/journal.pcbi.1008281.

Chambers, John M. 2014. "Object-Oriented Programming, Functional Programming and R." *Statistical Science* 29 (2). https://doi.org/10.1214/13-sts452.

Chessel, Anatole. 2017. "An Overview of Data Science Uses in Bioimage Informatics." *Methods* 115 (February): 110–18. https://doi.org/10.1016/j.ymeth.2016.12.014.

Corporation, Microsoft, and Steve Weston. 2011. "doParallel: Foreach Parallel Adaptor for the 'Parallel' Package." *CRAN: Contributed Packages*. The R Foundation. https://doi.org/10.32614/cran.package.doparallel.

Cosentino, Valerio, Javier Luis, and Jordi Cabot. 2016. "Findings from GitHub: Methods, Datasets and Limitations." In *Proceedings of the 13th International Conference on Mining Software Repositories*. ICSE '16. ACM. https://doi.org/10.1145/2901739.2901776.

Csardi, Gabor. 2016. "Cyclocomp: Cyclomatic Complexity of R Code." *CRAN: Contributed Packages*. The R Foundation. https://doi.org/10.32614/cran.package.cyclocomp.

Daniel Adler, Oleg Nenadic, Christian Gläser. 2007. "Ff: Memory-Efficient Storage of Large Data on Disk and Fast Access Functions." *CRAN: Contributed Packages*. The R Foundation. https://doi.org/10.32614/cran.package.ff.

Ding, Ning, Di An, Xiufeng Yin, and Yingxue Sun. 2020. "Detection and Evaluation of Microbeads and Other Microplastics in Wastewater Treatment Plant Samples." *Environmental Science and Pollution Research* 27 (13): 15878–87. https://doi.org/10.1007/s11356-020-08127-2.

Dinter, Franziska, Thomas Thiehle, Uwe Schedler, Werner Lehmann, Peter Schierack, and Stefan Rödiger. 2023. "Immobilisation of Lipophilic and Amphiphilic Biomarker on Hydrophobic Microbeads," January. https://doi.org/10.1101/2023.01.10.523433.

Ebert, Christof, James Cain, Giuliano Antoniol, Steve Counsell, and Phillip Laplante. 2016. "Cyclomatic Complexity." *IEEE Software* 33 (6): 27–29. https://doi.org/10.1109/ms.2016.147.

Ecke, Annemarie, Anne-Helen Lutter, Jenny Scholka, Anna Hansch, Roland Becker, and Ursula Anderer. 2019. "Tissue Specific Differentiation of Human Chondrocytes Depends on Cell Microenvironment and Serum Selection." *Cells* 8 (8): 934. https://doi.org/10.3390/cells8080934.

Eliceiri, Kevin W, Michael R Berthold, Ilya G Goldberg, Luis Ibáñez, B S Manjunath, Maryann E Martone, Robert F Murphy, et al. 2012. "Biological Imaging Software Tools." *Nature Methods* 9 (7): 697–710. https://doi.org/10.1038/nmeth.2084.

Fisher, R. A. 1992. "Statistical Methods for Research Workers." In *Breakthroughs in Statistics*, 66–70. Springer New York. https://doi.org/10.1007/978-1-4612-4380-9_6.

Folk, Mike, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. 2011. "An Overview of the HDF5 Technology Suite and Its Applications." In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*. EDBT/ICDT '11. ACM. https://doi.org/10.1145/1966895.1966900.

Geithe, Christiane, Bo Zeng, Carsten Schmidt, Franziska Dinter, Dirk Roggenbuck, Werner Lehmann, Gregory Dame, Peter Schierack, Katja Hanack, and Stefan Rödiger. 2021. "A Multiplex Microchamber Diffusion Assay for the Antibody-Based Detection of microRNAs on Randomly Ordered Microbeads," March. https://doi.org/10.1101/2021.03.19.436219.

———. 2024. "A Multiplex Microchamber Diffusion Assay for the Antibody-Based Detection of microRNAs on Randomly Ordered Microbeads." *Biosensors and Bioelectronics: X* 18 (June): 100484. https://doi.org/10.1016/j.biosx.2024.100484.

Gentleman, Robert, and Duncan Temple Lang. 2007. "Statistical Analyses and Reproducible Re-

search." *Journal of Computational and Graphical Statistics* 16 (1): 1–23. https://doi.org/10.1198/106186007X178663.

Ghosh, Swarnendu, Nibaran Das, Ishita Das, and Ujjwal Maulik. 2019. "Understanding Deep Learning Techniques for Image Segmentation." *ACM Computing Surveys* 52 (4): 1–35. https://doi.org/10.1145/3329784.

Giorgi, Federico M., Carmine Ceraolo, and Daniele Mercatelli. 2022. "The R Language: An Engine for Bioinformatics and Data Science." *Life* 12 (5): 648. https://doi.org/10.3390/life12050648.

Göröcs, Zoltán, Euan McLeod, and Aydogan Ozcan. 2015. "Enhanced Light Collection in Fluorescence Microscopy Using Self-Assembled Micro-Reflectors." *Scientific Reports* 5 (1). https://doi.org/10.1038/srep10999.

Gregory, Peggy. 2021. *Agile Processes in Software Engineering and Extreme Programming: 22nd International Conference on Agile Software Development, XP 2021, Virtual Event, June 14-18, 2021, Proceedings*. Edited by Casper Lassenius, Xiaofeng Wang, and Philippe Kruchten. Lecture Notes in Business Information Processing Ser. v.419. Cham: Springer International Publishing AG.

Haase, Robert, Elnaz Fazeli, David Legland, Michael Doube, Siân Culley, Ilya Belevich, Eija Jokitalo, Martin Schorb, Anna Klemm, and Christian Tischer. 2022. "A Hitchhiker's Guide Through the Bio-Image Analysis Software Universe." *FEBS Letters* 596 (19): 2472–85. https://doi.org/10.1002/1873-3468.14451.

Haralick, Robert M., K. Shanmugam, and Its'Hak Dinstein. 1973. "Textural Features for Image Classification." *IEEE Transactions on Systems, Man, and Cybernetics* SMC-3 (6): 610–21. https://doi.org/10.1109/tsmc.1973.4309314.

Hester, Jim. 2021. "GitHub Actions for the R language." https://github.com/r-lib/actions.

Hsu, Henry, and Peter A. Lachenbruch. 2014. "Paired t Test." *Wiley StatsRef: Statistics Reference Online*, September. https://doi.org/10.1002/9781118445112.stat05929.

Iglewicz, Boris, and David C. Hoaglin. 1993. *How to Detect and Handle Outliers*. The @ASQC Basic References in Quality Control: Statistical Techniques 16. Milwaukee, Wis.: ASQC Quality Press.

Jähne, Bernd, ed. 2002. *Digital Image Processing*. [CD-ROM-Ausg. der] 5., überarb. und erw. [gedr.] Aufl. Engineering Online Library. Berlin: Springer.

Jiang, Jiahui, Xiaoyu Cai, Hongyu Ren, Guangli Cao, Jia Meng, Defeng Xing, Jes Vollertsen, and Bingfeng Liu. 2024. "Effects of Polyethylene Terephthalate Microplastics on Cell Growth, Intracellular Products and Oxidative Stress of Scenedesmus Sp." *Chemosphere* 348 (January): 140760. https://doi.org/10.1016/j.chemosphere.2023.140760.

Kane, Michael J., John W. Emerson, Peter Haverty, and Charles Determan. 2008. "Bigmemory: Manage Massive Matrices with Shared Memory and Memory-Mapped Files." *CRAN: Contributed Packages*. The R Foundation. https://doi.org/10.32614/cran.package.bigmemory.

Knuth, D. E. 1984. "Literate Programming." *The Computer Journal* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.

Kohonen, T. 1990. "The Self-Organizing Map." *Proceedings of the IEEE* 78 (9): 1464–80. https://doi.org/10.1109/5.58325.

———. 2013. "Essentials of the Self-Organizing Map." *Neural Networks* 37 (January): 52–65. https://doi.org/10.1016/j.neunet.2012.09.018.

Koranne, Sandeep. 2011. *Handbook of Open Source Tools*. 1st ed. SpringerLink. Boston, MA: Springer US.

Lanubile, F., C. Ebert, R. Prikladnicki, and A. Vizcaino. 2010. "Collaboration Tools for Global Software Engineering." *IEEE Software* 27 (2): 52–55. https://doi.org/10.1109/ms.2010.39.

Liljequist, David, Britt Elfving, and Kirsti Skavberg Roaldsen. 2019. "Intraclass Correlation – a Discussion and Demonstration of Basic Features." Edited by Ferdinando Chiacchio. *PLOS ONE* 14 (7): e0219854. https://doi.org/10.1371/journal.pone.0219854.

Löfstedt, Tommy, Patrik Brynolfsson, Thomas Asklund, Tufve Nyholm, and Anders Garpebring. 2019. "Gray-Level Invariant Haralick Texture Features." Edited by Mathieu Hatt. *PLOS ONE* 14 (2): e0212110. https://doi.org/10.1371/journal.pone.0212110.

Maechler, Martin, Peter Rousseeuw, Anja Struyf, and Mia Hubert. 1999. "Cluster: 'Finding Groups in Data': Cluster Analysis Extended Rousseeuw Et Al." *CRAN: Contributed Packages*. The R Foundation. https://doi.org/10.32614/cran.package.cluster.

Matias, Filipe Inácio, Maria V. Caraza-Harter, and Jeffrey B. Endelman. 2020. "FIELDimageR: An R Package to Analyze Orthomosaic Images from Agricultural Field Trials." *The Plant Phenome Journal* 3 (1). https://doi.org/10.1002/ppj2.20005.

Matthias Gamer, Ian Fellows Puspendra Singh, Jim Lemon. 2005. "Irr: Various Coefficients of Interrater Reliability and Agreement." *CRAN: Contributed Packages*. The R Foundation. https://doi.org/10.32614/cran.package.irr.

McCabe, T. J. 1976. "A Complexity Measure." *IEEE Transactions on Software Engineering* SE-2 (4): 308–20. https://doi.org/10.1109/tse.1976.233837.

Meyer, Mathias. 2014. "Continuous Integration and Its Tools." *IEEE Software* 31 (3): 14–16. https://doi.org/10.1109/ms.2014.58.

Miller, Jeff. 1991. "Short Report: Reaction Time Analysis with Outlier Exclusion: Bias Varies with Sample Size." *The Quarterly Journal of Experimental Psychology Section A* 43 (4): 907–12. https://doi.org/10.1080/14640749108400962.

Mittal, Mamta, Amit Verma, Iqbaldeep Kaur, Bhavneet Kaur, Meenakshi Sharma, Lalit Mohan Goyal, Sudipta Roy, and Tai-Hoon Kim. 2019. "An Efficient Edge Detection Approach to Provide Better Edge Connectivity for Image Analysis." *IEEE Access* 7: 33240–55. https://doi.org/10.1109/access.2019.2902579.

Moen, Erick, Dylan Bannon, Takamasa Kudo, William Graf, Markus Covert, and David Van Valen. 2019. "Deep Learning for Cellular Image Analysis." *Nature Methods* 16 (12): 1233–46. https://doi.org/10.1038/s41592-019-0403-1.

Morel, Jean-Michel, Ana-Belen Petro, and Catalina Sbert. 2014. "Screened Poisson Equation for Image Contrast Enhancement." *Image Processing On Line* 4 (March): 16–29. https://doi.org/10.5201/ipol.2014.84.

Mouselimis, Lampros. 2016. "OpenImageR: An Image Processing Toolkit." *CRAN: Contributed Packages*. The R Foundation. https://doi.org/10.32614/cran.package.openimager.

Müller, Kirill, and Lorenz Walthert. 2017. "Styler: Non-Invasive Pretty Printing of R Code." *CRAN: Contributed Packages*. The R Foundation. https://doi.org/10.32614/cran.package.styler.

Murphy, Robert F. 2014. "A New Era in Bioimage Informatics." *Bioinformatics* 30 (10): 1353–53. https://doi.org/10.1093/bioinformatics/btu158.

Myers, Glenford J. 2012. *The Art of Software Testing*. Edited by Corey Sandler and Tom Badgett. 3rd ed. Hoboken, N.J.: J. Wiley & Sons.

Niedballa, Jürgen, Jan Axtner, Timm Fabian Döbert, Andrew Tilker, An Nguyen, Seth T. Wong, Christian Fiderer, Marco Heurich, and Andreas Wilting. 2022. "Imageseg: An R Package for Deep Learning-Based Image Segmentation." *Methods in Ecology and Evolution* 13 (11): 2363–71. https://doi.org/10.1111/2041-

210x.13984.

Ochi, Shota. 2018. "imagerExtra: Extra Image Processing Library Based on 'Imager'." *CRAN: Contributed Packages*. The R Foundation. https://doi.org/10.32614/cran.package.imagerextra.

Olivoto, Tiago. 2022. "Lights, Camera, Pliman! An R Package for Plant Image Analysis." *Methods in Ecology and Evolution* 13 (4): 789–98. https://doi.org/10.1111/2041-210x.13803.

Ooms, Jeroen. 2016. "Magick: Advanced Graphics and Image-Processing in R." *CRAN: Contributed Packages*. The R Foundation. https://doi.org/10.32614/cran.package.magick.

Pau, Grégoire, Florian Fuchs, Oleg Sklyar, Michael Boutros, and Wolfgang Huber. 2010. "EBImage—an R Package for Image Processing with Applications to Cellular Phenotypes." *Bioinformatics* 26 (7): 979–81. https://doi.org/10.1093/bioinformatics/btq046.

Paul-Gilloteaux, Perrine. 2023. "Bioimage Informatics: Investing in Software Usability Is Essential." *PLOS Biology* 21 (7): e3002213. https://doi.org/10.1371/journal.pbio.3002213.

Peng, Hanchuan. 2008. "Bioimage Informatics: A New Area of Engineering Biology." *Bioinformatics* 24 (17): 1827–36. https://doi.org/10.1093/bioinformatics/btn346.

Peng, Hanchuan, Alex Bateman, Alfonso Valencia, and Jonathan D. Wren. 2012. "Bioimage Informatics: A New Category in Bioinformatics." *Bioinformatics* 28 (8): 1057–57. https://doi.org/10.1093/bioinformatics/bts111.

Peng, Roger D. 2022. *R Programming for Data Science*. Leanpub. https://bookdown.org/rdpeng/rprogdatascience/.

Peng, Roger D., Sean Kross, and Brooke Anderson. 2016. *Mastering-Software-Development-in-R/Mastering Software Development in R (Book).pdf at Master · Simbosky/Mastering-Software-Development-in-R. GitHub*. Leanpub. https://github.com/simbosky/Mastering-Software-Development-in-R/blob/master/Mastering%20Software%20Development%20in%20R%20(book).pdf.

Perez-Riverol, Yasset, Laurent Gatto, Rui Wang, Timo Sachsenberg, Julian Uszkoreit, Felipe da Veiga Leprevost, Christian Fufezan, et al. 2016. "Ten Simple Rules for Taking Advantage of Git and GitHub." Edited by Scott Markel. *PLOS Computational Biology* 12 (7): e1004947. https://doi.org/10.1371/journal.pcbi.1004947.

Prajapati, Vignesh. 2013. *Big Data Analytics with R and Hadoop*. Online-Ausg. Birmingham: Packt Publishing.

Prewitt, Judith MS et al. 1970. "Object Enhancement and Extraction." *Picture Processing and Psychopictorics* 10 (1): 15–19.

R Core Team. 2024. "Writing R Extensions." https://cran.r-project.org/doc/manuals/R-exts.html.

Reddig, Annika, Claudia E. Rübe, Stefan Rödiger, Peter Schierack, Dirk Reinhold, and Dirk Roggenbuck. 2018. "DNA Damage Assessment and Potential Applications in Laboratory Diagnostics and Precision Medicine." *Journal of Laboratory and Precision Medicine* 3 (4): 31–31. https://doi.org/10.21037/jlpm.2018.03.06.

Reynolds, A. P., G. Richards, B. de la Iglesia, and V. J. Rayward-Smith. 2006. "Clustering Rules: A Comparison of Partitioning and Hierarchical Clustering Algorithms." *Journal of Mathematical Modelling and Algorithms* 5 (4): 475–504. https://doi.org/10.1007/s10852-005-9022-1.

Rigo, Norbert, Chengfu Sun, Patrizia Fabrizio, Berthold Kastner, and Reinhard Lührmann. 2015. "Protein Localisation by Electron Microscopy Reveals the Architecture of the Yeast Spliceosomal b Complex." *The EMBO Journal* 34 (24): 3059–73. https://doi.org/10.15252/embj.201592022.

Rivest, R. 1992. "The MD5 Message-Digest Algorithm." Request for Comments. RFC 1321; RFC Editor.

https://doi.org/10.17487/RFC1321.

Roberts, Lawrence G. 1980. *Machine Perception of Three-Dimensional Solids*. Garland Pub.

Rödiger, Stefan, Michał Burdukiewicz, Konstantin A. Blagodatskikh, and Peter Schierack. 2015. "R as an Environment for the Reproducible Analysis of DNA Amplification Experiments." *The R Journal* 7 (2): 127–50. https://doi.org/10.32614/RJ-2015-011.

Rödiger, Stefan, Claudia Liebsch, Carsten Schmidt, Werner Lehmann, Ute Resch-Genger, Uwe Schedler, and Peter Schierack. 2014. "Nucleic Acid Detection Based on the Use of Microbeads: A Review." *Microchimica Acta* 181 (11–12): 1151–68. https://doi.org/10.1007/s00604-014-1243-4.

Rödiger, Stefan, Marius Liefold, Madeleine Ruhe, Mark Reinwald, Eberhard Beck, and P. Markus Deckert. 2018. "Quantification of DNA Double-Strand Breaks in Peripheral Blood Mononuclear Cells from Healthy Donors Exposed to Bendamustine by an Automated γH2AX Assay—an Exploratory Study." *Journal of Laboratory and Precision Medicine* 3 (May): 47–47. https://doi.org/10.21037/jlpm.2018.04.10.

Rousseeuw, Peter J. 1987. "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis." *Journal of Computational and Applied Mathematics* 20 (November): 53–65. https://doi.org/10.1016/0377-0427(87)90125-7.

Ruhe, Madeleine, Dominik Rabe, Christoph Jurischka, Julia Schröder, Peter Schierack, P. Markus Deckert, and Stefan Rödiger. 2019. "Molecular Biomarkers of DNA Damage in Diffuse Large-Cell Lymphoma—a Review." *Journal of Laboratory and Precision Medicine* 4 (5): 1–20. https://doi.org/10.21037/jlpm.2019.01.01.

Scharr, Hanno. 2000. "Optimale Operatoren in Der Digitalen Bildverarbeitung." https://doi.org/10.11588/HEIDOK.00000962.

Schmidberger, Markus, Martin Morgan, Dirk Eddelbuettel, Hao Yu, Luke Tierney, and Ulrich Mansmann. 2009. "State-of-the-Art in Parallel Computing with R." https://doi.org/10.5282/UBM/EPUB.8991.

Schneider, Jens, Romano Weiss, Madeleine Ruhe, Tobias Jung, Dirk Roggenbuck, Ralf Stohwasser, Peter Schierack, and Stefan Rödiger. 2019. "Open Source Bioimage Informatics Tools for the Analysis of DNA Damage and Associated Biomarkers." *Journal of Laboratory and Precision Medicine* 4 (0): 1–27. https://doi.org/10.21037/jlpm.2019.04.05.

Schubert, Erich, and Peter J. Rousseeuw. 2019. "Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms." In *Lecture Notes in Computer Science*, 171–87. Springer International Publishing. https://doi.org/10.1007/978-3-030-32047-8_16.

Seo, Songwon. 2006. "A Review and Comparison of Methods for Detecting Outliers in Univariate Data Sets." http://d-scholarship.pitt.edu/7948/.

Shapiro, S. S., and M. B. Wilk. 1965. "An Analysis of Variance Test for Normality (Complete Samples)." *Biometrika* 52 (3–4): 591–611. https://doi.org/10.1093/biomet/52.3-4.591.

Siddik, Abu Bucker, Steven Sandoval, David Voelz, Laura E. Boucheron, and Luis Varela. 2023. "Deep Learning Estimation of Modified Zernike Coefficients and Recovery of Point Spread Functions in Turbulence." *Optics Express* 31 (14): 22903. https://doi.org/10.1364/oe.493229.

Soares, Eliezio, Gustavo Sizilio, Jadson Santos, Daniel Alencar da Costa, and Uirá Kulesza. 2022. "The Effects of Continuous Integration on Software Development: A Systematic Literature Review." *Empirical Software Engineering* 27 (3). https://doi.org/10.1007/s10664-021-10114-1.

Sobel, Irwin. 2014. "An Isotropic 3x3 Image Gradient Operator." *Presentation at Stanford A.I. Project 1968*, February.

Song, Feijun, Qiao Chen, Xiongxin Tang, and Fanjiang Xu. 2024. "Analytical Model of Point Spread Func-

tion Under Defocused Degradation in Diffraction-Limited Systems: Confluent Hypergeometric Function." *Photonics* 11 (5): 455. https://doi.org/10.3390/photonics11050455.

Sonka, Milan, and J. Michael Fitzpatrick, eds. 2000. *Handbook of Medical Imaging.* SPIE Press Monograph PM80. Bellingham, Wash.: SPIE Press.

Spinellis, Diomidis. 2012. "Git." *IEEE Software* 29 (3): 100–101. https://doi.org/10.1109/ms.2012.61.

Swedlow, Jason R., and Kevin W. Eliceiri. 2009. "Open Source Bioimage Informatics for Cell Biology." *Trends in Cell Biology* 19 (11): 656–60. https://doi.org/10.1016/j.tcb.2009.08.007.

Swedlow, Jason R., Ilya G. Goldberg, and Kevin W. Eliceiri. 2009. "Bioimage Informatics for Experimental Biology." *Annual Review of Biophysics* 38 (1): 327–46. https://doi.org/10.1146/annurev.biophys.050708.133641.

Sydor, Andrew M., Kirk J. Czymmek, Elias M. Puchner, and Vito Mennella. 2015. "Super-Resolution Microscopy: From Single Molecules to Supramolecular Assemblies." *Trends in Cell Biology* 25 (12): 730–48. https://doi.org/10.1016/j.tcb.2015.10.004.

Takashimizu, Yasuhiro, and Maiko Iiyoshi. 2016. "New Parameter of Roundness R: Circularity Corrected by Aspect Ratio." *Progress in Earth and Planetary Science* 3 (1). https://doi.org/10.1186/s40645-015-0078-x.

V, Bino Sebastian. 2012. "Grey Level Co-Occurrence Matrices: Generalisation and Some New Features." *International Journal of Computer Science, Engineering and Information Technology* 2 (2): 151–57. https://doi.org/10.5121/ijcseit.2012.2213.

Vassilev, Boris, Riku Louhimo, Elina Ikonen, and Sampsa Hautaniemi. 2016. "Language-Agnostic Reproducible Data Analysis Using Literate Programming." Edited by Frederique Lisacek. *PLOS ONE* 11 (10): e0164023. https://doi.org/10.1371/journal.pone.0164023.

Vidoni, Melina. 2021. "Evaluating Unit Testing Practices in R Packages." In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. https://doi.org/10.1109/icse43902.2021.00136.

Vuorre, Matti, and James P. Curley. 2018. "Curating Research Assets: A Tutorial on the Git Version Control System." *Advances in Methods and Practices in Psychological Science* 1 (2): 219–36. https://doi.org/10.1177/2515245918754826.

Wehrens, Ron, and Johannes Kruisselbrink. 2006. "Kohonen: Supervised and Unsupervised Self-Organising Maps." *CRAN: Contributed Packages.* The R Foundation. https://doi.org/10.32614/cran.package.kohonen.

———. 2018. "Flexible Self-Organizing Maps in Kohonen 3.0." *Journal of Statistical Software* 87 (7). https://doi.org/10.18637/jss.v087.i07.

Weston, Steve. 2009. "Foreach: Provides Foreach Looping Construct." *CRAN: Contributed Packages.* The R Foundation. https://doi.org/10.32614/cran.package.foreach.

Wickham, Hadley. 2009. "Testthat: Unit Testing for R." *CRAN: Contributed Packages.* The R Foundation. https://doi.org/10.32614/cran.package.testthat.

———. 2011. "Testthat: Get Started with Testing." *The R Journal* 3 (1): 5. https://doi.org/10.32614/rj-2011-002.

———. 2023. *R Packages: Organize, Test, Document, and Share Your Code.* Edited by Jenny Bryan. Second edition. Beijing: O'Reilly.

Zhang, Jing, Swati Shikha, Qingsong Mei, Jinliang Liu, and Yong Zhang. 2019. "Fluorescent Microbeads for Point-of-Care Testing: A Review." *Microchimica Acta* 186 (6). https://doi.org/10.1007/s00604-019-3449-y.

Zhu, Hong, Patrick A. V. Hall, and John H. R. May. 1997. "Software Unit Test Coverage and Adequacy." *ACM Computing Surveys* 29 (4): 366–427. https://doi.org/10.1145/267580.267590.

ZJ, Dai, and Jacky Poon. 2019. "Disk.frame: Larger-Than-RAM Disk-Based Data Manipulation Framework." *CRAN: Contributed Packages*. The R Foundation. https://doi.org/10.32614/cran.package.disk.frame.