

CHAPTER 7

Object measurements from 2D microscopy images

Peter Bajcsy*, Joe Chalfoun*, Mylene Simon*, Marcin Kociolek[†], and Mary Brady*

*Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, United States

[†]Lodz University of Technology, Lodz, Poland

Contents

1. Background	159
2. Introduction to 2D image measurements	161
3. Approach to numerical evaluations of feature variability and feature-based classification	164
4. Integration of open-source libraries for 2D image measurements	165
4.1 Integration of feature extraction libraries	166
4.2 Summary of features in all integrated libraries	167
5. Image features in scientific use cases	168
6. Variability of image features	170
6.1 Feature variability metric	170
6.2 Image feature variability analysis	171
6.3 Sources of image feature variations	171
6.4 Discussion	176
7. Feature-based classification	177
7.1 Classification variability metric	178
7.2 Classification variability analysis	178
7.3 Discussion	178
8. Summary	179
Appendix: Online information about the work	181
Acknowledgments	181
Disclaimer	181
References	181

1. Background

Two-dimensional (2D) image measurements are an integral part of quantitative microscopy imaging and image informatics. Such measurements are computed over a set of connected image pixels (region of interest, or ROI) using a wide variety of mathematical operations applied to pixel location and intensity values. Numerical results of these computations are frequently denoted as image or object features. To compute a 2D object feature, one needs the following:

- (1) An input digital image
- (2) A mask image defining each ROI by a unique label assigned to a set of connected pixels

- (3) Mathematical operations applied to the pixels forming a ROI
- (4) Parameters for the mathematical operations (model).

Every image can yield very many extracted features, as the combinatorial number of ROIs, mathematical operations, and their parameters can be extremely large. As the imaging field advances, the object feature extraction faces challenges not only in the number of measurements per image, but also in the number of images that must be processed for each experiment. To address the issue of an extremely large number of feature extractions in quantitative imaging, object measurements from 2D microscopy images have been automated by using software. As a consequence, automation has led to the creation of many software libraries for feature extraction.

For example, the microscopy community has been using several open-source libraries, including the Python scikit-image [1], CellProfiler [2], MaZda [3], ImageJ/Fiji [4], and WND-CHARM [5], and feature extractors built in house using open-source libraries (e.g., OpenCV [6], Java ImageIO [7]) and commercial libraries (e.g., MATLAB, Wolfram Mathematica). The use of some of these libraries in scholarly publications is illustrated in Table 7.1. The table was created based on Google Scholar queries and documents the usage of intensity, shape, and textural image features extracted by using one of the four libraries. The large counts in Table 7.1 motivate us to study the feature variability across feature extraction libraries, as the scientific conclusions derived here might be biased by the choice of library.

Given this motivation, the goal of this chapter is to characterize object feature variability across software libraries and to demonstrate the impact of such a numerical variability on feature-based classification. By analyzing feature variability across software libraries, one can answer the following questions for scientists:

- (1) What image features do not vary with software implementations in open-source software libraries?

Table 7.1 Total counts of scholarly publications published since 2013 and since 2017 until 2018

Google scholar query key phrases	Since 2017	Since 2013
“CellProfiler image intensity features”	482	1860
“MaZda software image intensity features”	233	912
“Python scikit-image intensity features”	764	1800
“MATLAB image intensity features”	12,200	19,900
“CellProfiler shape features”	406	1590
“MaZda software shape features”	312	1600
“Python scikit-image shape features”	1760	4110
“MATLAB shape features”	17,800	29,300
“CellProfiler texture features”	104	484
“MaZda software texture features”	129	489
“Python scikit-image texture features”	385	942
“MATLAB texture features”	5900	19,400

- (2) What is the ranking of features based on the magnitude of their numerical variability?
- (3) What are the sources of numerical feature variability?
- (4) How much does feature variability affect feature-based classification outcomes?

We are not answering these questions related to the absolute accuracy of feature values because we are not creating any evaluation ground truths. Our goal is to quantify the relative feature differences and investigate the sources of those differences in order to emphasize the importance of gathering computational provenance information and noting possible biases in scholarly publications based on the selection of a particular feature extraction library.

This chapter tackles the following goals:

- Introduce 2D image feature extraction and classification.
- Describe an overall approach to numerical evaluation of feature variability and classification.
- Outline the integration of open-source software implementations of 2D image feature extractors.
- Present representative microscopy images collected to extract features determining the heterogeneity of stem cell colonies from terabyte (TB)-sized videos [8, 9].
- Quantify and explain sources of numerical variability of image features across image feature extraction libraries.
- Illustrate the impact of the feature variability on classification outcomes in the context of a biological imaging experiment.

In addition to the insights provided in this chapter, we have prepared an open-source client-server software system called a *web image processing pipeline (WIPP)* [10], which integrates multiple feature extraction libraries. It is available for download at <https://isg.nist.gov>. By reading this chapter and using WIPP, scientists can evaluate feature variability across software libraries on their own. Scientists can also use WIPP for gathering provenance information about their feature extractions to make their work reproducible.

2. Introduction to 2D image measurements

Many image feature extractors have been implemented in academic environments [5, 11, 12], commercial platforms [13], and publicly available image libraries [1, 4, 6]. These extractors generate image features that are used primarily for (1) classification, (2) discovery in scientific pursuits, or both. In the context of feature-based classification, one is searching for the most discriminative or predictive features considering a certain number of classes. In the context of discovery, one is trying to understand statistical and semantic object characteristics based on image features.

In general, feature extractors can provide:

- (1) Low-level image descriptors, such as scale-invariant feature transform (SIFT) [14], speeded-up robust features (SURFs) [15], histograms of oriented gradients

- (HOGs) [16], local phase quantization (LPQ), binarized statistical image features (BSIFs) [17], local binary patterns (LBPs) [18], and local ternary patterns (LTPs) [19].
- (2) High-level object descriptors (e.g., average intensity, location, size, perimeter, change of location).

We are interested in high-level object descriptors because they correspond to image-based measurements of semantically meaningful objects. Nonetheless, a computation of low-level features might be part of the computation to derive a high-level descriptor. Computations of high-level descriptors might also include image transformations. For example, Fourier transform (FT) changes the semantic meaning of the underlying Cartesian coordinate space of raster images to represent periodic structures for direct object measurements.

High-level object features can describe the spectral, spatial, textural, or temporal properties of an object. Spectral intensity features are typically statistical measurements of a probability distribution computed from pixel intensity (i.e., central moments). Similarly, spatial shape features are spatial moments derived from coordinates of pixels included in one ROI. Intensity and shape features have relatively well defined mapping between visual perception and their mathematical formula. On the other hand, perception of image texture is only coarsely mapped to the mathematical formulas of textural features. Computations of textural features typically include some transformation to a space where periodically repeating patterns and scale-dependent primitives can be quantified (e.g., Texton theory [20]). Temporal motion features capture dynamic changes of object appearance and location properties, such as biological cell states (e.g., mitosis, differentiation, apoptosis, migration). While computations of all feature types require 2D ROIs, motion features also require tracking information between two time-adjacent images. The tracking information defines the correspondence between object identifiers in labeled masks at time t and at time $(t+1)$.

In addition to the feature types, there are many other ways to establish ROIs that represent semantically meaningful objects. ROIs can be defined as rectangular fields of view (FOVs), arbitrarily shaped ROIs resulting from image segmentation, grid squares or grid hexagons subdividing ROIs obtained from image segmentation, and many other space filling 2D shapes subdividing an image. Fig. 7.1 illustrates the options for extracting image features from a cell colony with or without a mask, as well as other suboptions. In our work, we assume that users create ROIs according to their application needs and provide them as input masks (labeled images).

The process of designing, extracting, and selecting features is called *feature engineering*. Feature engineering is a scientific problem on its own, which faces challenges in terms of the following:

- (a) Segmenting an image into semantically meaningful ROIs
- (b) Devising mathematical models for operating on pixels in each ROI
- (c) Selecting the most application-relevant features and computational parameters

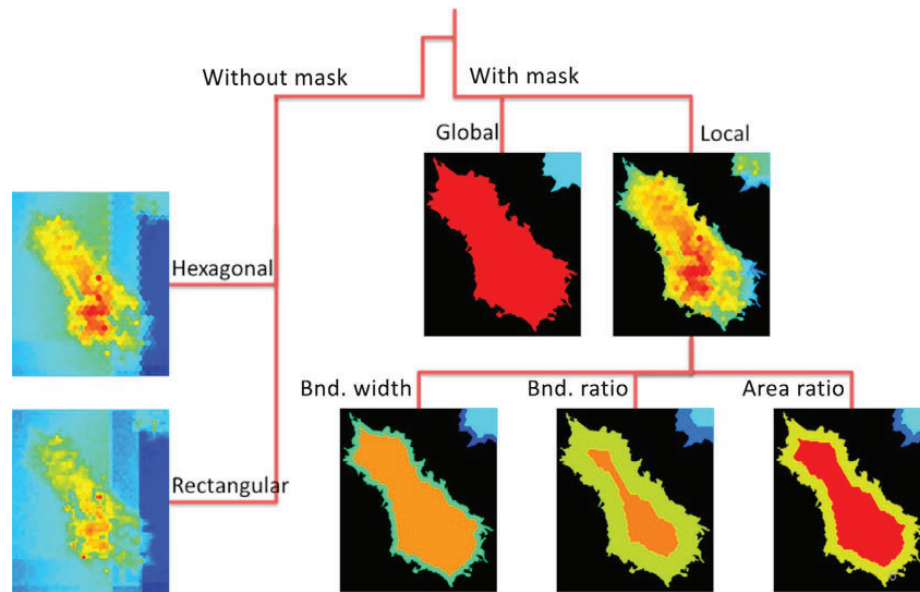


Fig. 7.1 Possible options to creating ROIs for computing 2D image features.

- (d) Selecting the feature-based classification model and model parameters
- (e) Optimizing all selections with respect to some reference data with known classification outcomes

If the end goal is to obtain the most accurate classification, then the feature engineering problem might be replaced by the problem of designing a convolutional neural network (CNN) architecture and the associated problem of creating a sufficiently large reference data set [21–24]. However, if the end goal is to gain insights into the characteristics of a phenomenon, then feature engineering remains a challenge [25, 26]. In our work, we consider the latter end goal and assume that a scientist has identified semantically meaningful ROIs and selected image measurements over ROIs. Our main concern is only the feature variability and its classification impact in the context of a feature engineering problem.

The feature variability across software libraries is of concern not only to the microscopy imaging community, but also to the medical imaging community. For example, Bhadiraju et al. [8] studied the reproducibility of quantitative imaging features used for classifying lung tumors in computed tomography (CT) images. The variability of features was determined independent of segmentation over repeated acquisitions of lung CT data sets. The study used commercial features implemented in C/C++ and MATLAB, which makes it hard to understand the sources of numerical variability. Thus, we limited our study to open-source feature extraction libraries.

The feature variability study can be related to a sensitivity study from a statistical perspective. Given all software implementation variables (factors), full factorial experimental designs would establish the sensitivity of the numerical features to the factors.

Such studies are frequently conducted with medical imaging devices (ultrasound, computer tomography, or magnetic resonance imaging) [10, 11]. Our feature variability study cannot be executed using a full factorial experimental design because many software libraries have hidden factors that cannot vary. However, our analysis can demonstrate the impact of feature variability on the classification of cell colonies that are heterogeneous from already-published stem cell microscopy images [5].

3. Approach to numerical evaluations of feature variability and feature-based classification

We approach numerical evaluations of image feature variability and its classification impact by executing the following steps:

- (1) Identify overlapping image features in multiple feature extraction libraries.
- (2) Integrate all feature extraction libraries into a unified software framework.
- (3) Select and upload data sets that yield a wide range of feature values.
- (4) Set all exposed algorithmic inputs to the same values.
- (5) Establish metrics of feature and classification variabilities.
- (6) Quantify numerical variability of intensity, shape, and textural features and classification outcomes.
- (7) Analyze sources of variability.

These steps are illustrated in Fig. 7.2 (feature variability) and Fig. 7.3 (classification variability), with an emphasis on the analysis flow while the feature extraction libraries are the only element in the flow varying during the numerical evaluations. As shown

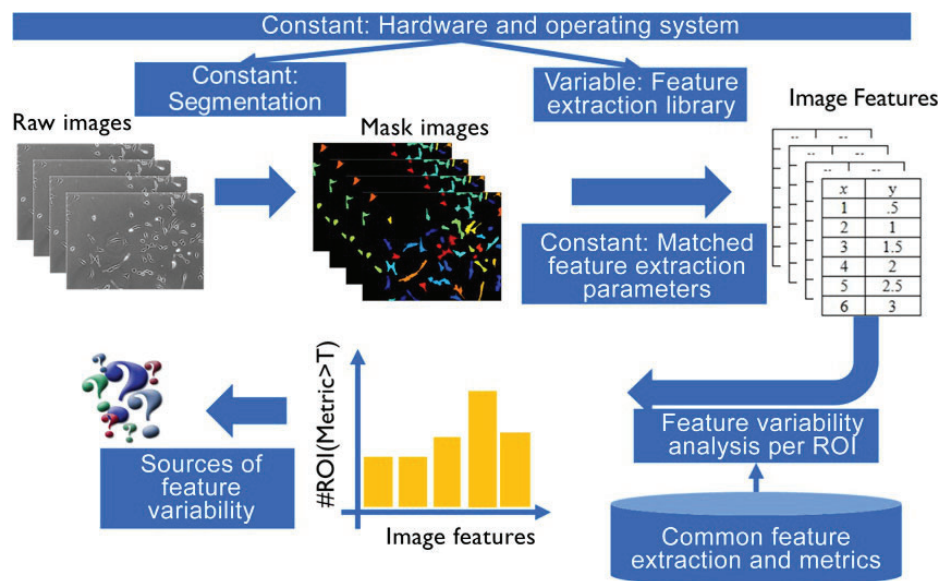


Fig. 7.2 Variability analysis of image features; arrows show analysis flow.

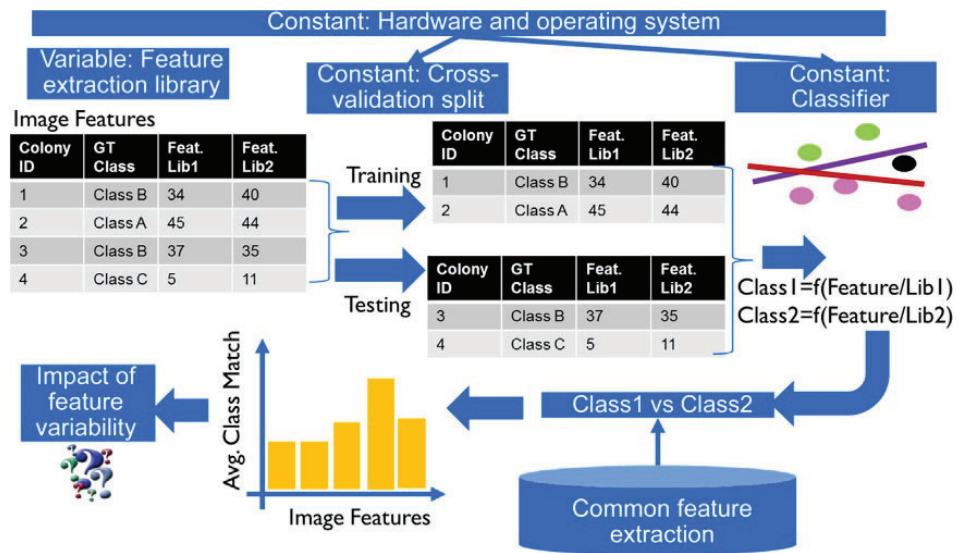


Fig. 7.3 Variability analysis of feature-based classification; arrows show analysis flow.

in Fig. 7.2, hardware, segmentation, and feature extraction parameters are kept constant. In Fig. 7.3, we indicated that hardware, cross-validation split, and classifier models are constant. Note that we evaluated the results from multiple nonstochastic classifiers.

We manually identified overlapping image features in multiple feature extraction libraries (step 1) by considering image features extracted using ImageJ/Fiji [4], Python scikit-image [1], CellProfiler [2, 12], MaZda [27–29], and feature extractors developed in-house using Java ImageIO [30]. Integration of all the feature extraction libraries to a WIPP framework [10] (step 2) allowed us to establish an identical computational environment, gather provenance information, and rerun feature extractors as many times as needed after test images were uploaded to the WIPP client-server system (step 3). These efforts are described in Section 4 (step 2) and in Section 5 (step 3). To set feature extraction algorithmic inputs, we identified exposed and hidden parameters, kept the default value of hidden parameters, and modified exposed parameters to comparable values with other feature extractors (step 4). The comparison metrics for feature and classification variabilities (step 5), together with the numerical results (step 6) and our insights about sources of numerical variability (step 7), are provided in Sections 6 and 7.

4. Integration of open-source libraries for 2D image measurements

We first integrated image feature extraction algorithms from open-source libraries into WIPP [10]. We unified the application programming interface (API) to feature extraction algorithms via a common Extensible Markup Language (XML) file with input and

output specifications (see Section 4.1). Next, we summarized all the features for each type and counted the number of common features per pairs of software libraries (see Section 4.2).

4.1 Integration of feature extraction libraries

Although integration of feature libraries was the main goal, we also had to consider the computational scalability and traceability of image feature executions. For this purpose, we leveraged past work on scientific workflow management systems because image features are computed as a sequence of computational steps. One of these steps is the execution of software found in a third-party feature extraction library with the specified inputs. To enable execution (i.e., integration of heterogeneous software), we predefine a common file interface for inputs, design a parser for input parameters in the programming language of each image library, and automate launching third-party software via a command-line interface. The execution of feature extractions is then scheduled, monitored, and managed by a scientific workflow system.

In an effort to provide access to traceable image features, we designed the WIPP architecture shown in Fig. 7.4. The main capabilities of this web-based framework for traceable image feature extraction are (1) extensibility to include image feature extraction libraries written in any programming language via a file interface, (2) data management to

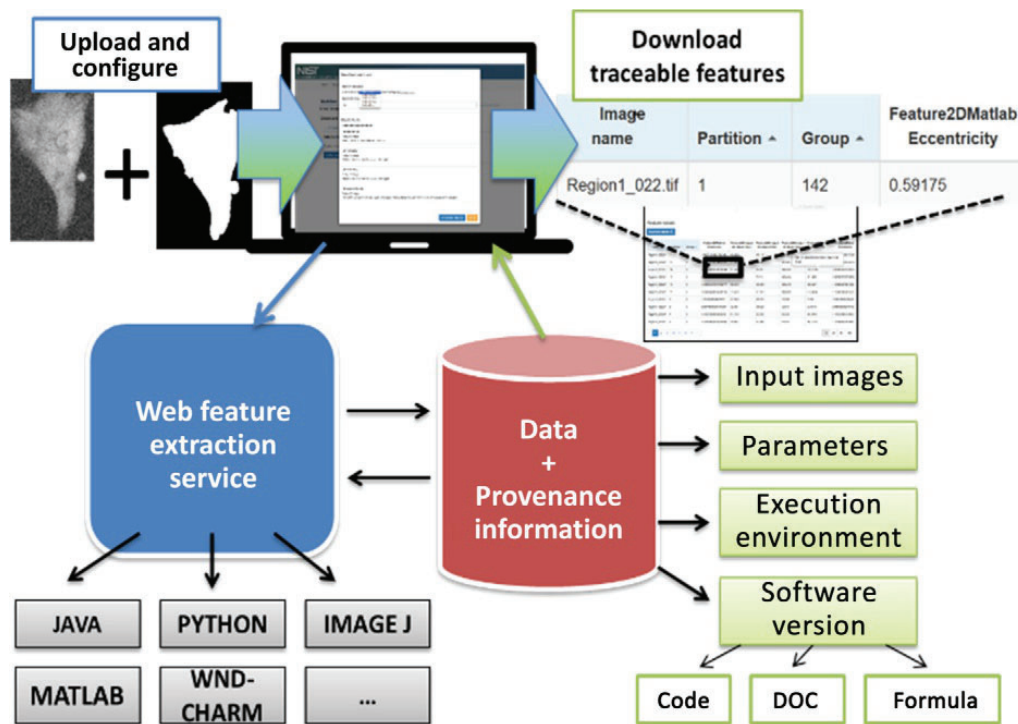


Fig. 7.4 Architecture of a client-server system for traceable image feature extraction.

upload collections and download feature values, (3) configuration and execution interface to feature extractors registered in the system, (4) collaborative access to traceable image feature values that are hyperlinked to their provenance information, and (5) access to all downloadable computational artifacts that are needed for reexecution.

In this client-server application, the server side has a representational state transfer (REST) or RESTful API, built using the Java Spring framework. It is coupled with the MongoDB database and a scientific workflow management system (WMS) called Pegasus [31] to manage the feature extraction jobs. The client side is a light web application written in JavaScript using the AngularJS framework. The client side communicates via the REST API with the server side.

4.2 Summary of features in all integrated libraries

We evaluated five open-source libraries of feature extractors that have been integrated in WIPP. Table 7.2 summarizes the number of overlapping features per software library and the total number of features per feature type. These five libraries offer extractions of 472 unique intensity, shape, and textural features. The split of 472 unique features across the five libraries is the Python scikit-image (45), CellProfiler (125), MaZda (192), Java (77), and ImageJ/Fiji (33). Of the total number of 472 features, there are 301 textural features that are split across the four libraries as follows: Python (24 out of 45), CellProfiler (53 out of 125), MaZda (172 out of 192), and in-house Java (52 out of 77). Features in Python were calculated on top of an existing image-processing library (scikit-image). The features in ImageJ/Fiji were computed as a plug-in using the ImageJ API [4], and Java features were implemented from scratch at the National Institute of Standards and Technology (NIST) [30]. MaZda and CellProfiler are stand-alone software libraries that have a defined API to access a variety of features.

Table 7.2 Integrated software libraries and their counts of the common intensity, shape, and textural features with other software libraries (#common) and of the total number of features (#total)

Feature type	Library count	Python	CellProfiler	MaZda	Java	ImageJ/Fiji
Intensity feature	#common	3	5	4	6	8
	#total	3	23	13	9	10
Shape feature	#common	18	11	3	12	14
	#total	18	49	7	16	23
Textural feature	#common	16	44	44	0	X
	#total	24	53	172	52	X

X—stands for not included in the analyses.

Note: Due to the manual work involved in integrating features, we have not included all features in all libraries. For example, the Python scikit-image and MaZda libraries contain more features than those integrated in WIPP for quantitative comparison.

5. Image features in scientific use cases

To illustrate numerical variations of intensity, shape, and textural features, as well as the classification impact, we chose data from a published imaging experiment [8]. The ultimate goal of this experiment was to measure the presence and classify the distribution of Oct4 pluripotency markers in living cell colonies over a large field of view. The image processing in this experiment is based on extracting colony features and then classifying them based on those features. The use of object features in this experiment is common to many other published microscopy experiments (e.g., cancer diagnosis from immunohistochemistry images [32]).

As documented in Bhadriraju et al. [8], the images of cell colonies were taken every 45 min for 5 days (161 time points) in both phase-contrast and fluorescent-imaging modalities. At every time point, one grid of 16×22 individual fields of view (FOVs) was taken with a 10% overlap between FOVs in both the x - and y -directions. A subset of all cell colonies was manually classified into three categories (homogeneous, heterogeneous, and dark), based on the distribution of the Oct4 marker in each colony. The class labels correspond to the visual assessment of Oct4 intensities within each cell colony. The *homogeneous* label refers to the case when the majority of pixel intensities have a high value (white). The *dark* label refers to the majority of low pixel intensities (black) and the *heterogeneous* label refers to a mixture of pixel intensities.

To automate the processing of terabyte-sized images, the colony features were extracted from the fluorescent channel and the colony masks were obtained from the coregistered phase contrast channel. A gigapixel image at each time point was created by using published background correction [33], stitching [34], segmentation [35], tracking [36], and feature-extraction methods [37]. To predict classification labels, a supervised logistic regression model was developed using colony features extracted from 140 stem cell colonies that were manually categorized as homogeneous, heterogeneous, or dark. Fig. 7.5 illustrates colony examples for each classification category in both acquired imaging modalities [i.e., phase contrast and green fluorescent protein (GFP) channels].

The inputs into our study were as follows:

- (1) 140 manually annotated cell colonies from 161 stitched and background corrected fluorescent Oct4 images
- (2) The corresponding colony masks computed by segmenting the stitched phase contrast images
- (3) The classification labels assigned manually by experts

We opted to conduct the study with measured images rather than synthetic images because we had no mathematical models for generating synthetic images that span a wide range of image features, and we found the measured images to be a good initial approximation of the range for most of the image features. Figs. 7.6 and 7.7 illustrate the ranges

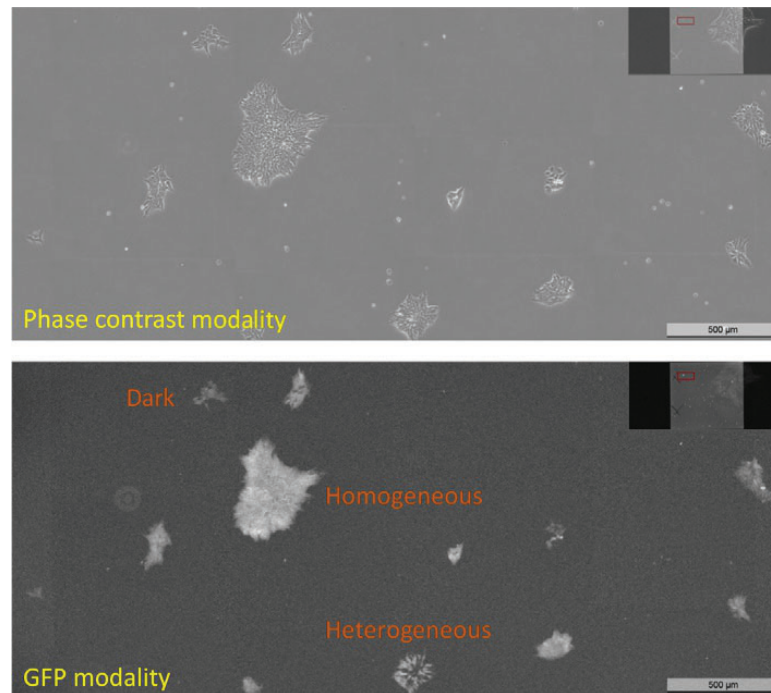


Fig. 7.5 Cell colony classification into three classes: homogeneous (bright intensities), heterogeneous (mixed bright and dark intensities), and dark intensity (indistinguishable dark intensities from background). The classification is based on gray-level image intensities of fluorescently stained cells with Oct4 pluripotency marker.

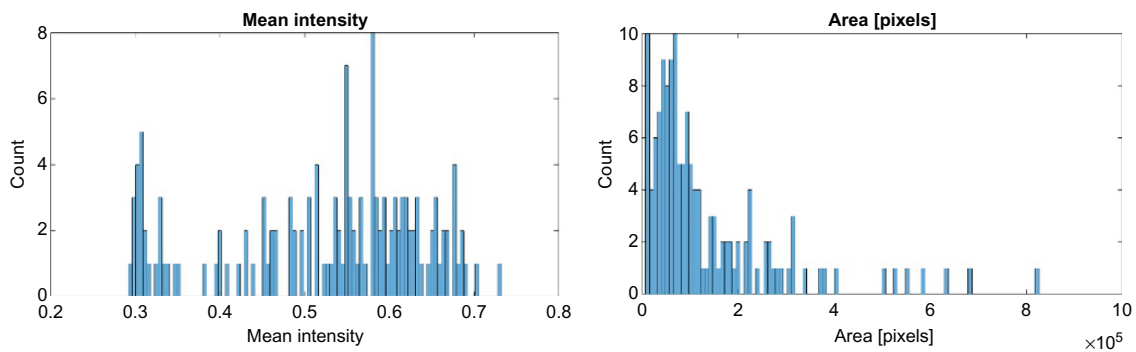


Fig. 7.6 Histograms of mean intensity (left) and area (right) features from the objects defined by test images and their masks.

of image features extracted from the selected measured images (140 colonies). Fig. 7.6 shows the histograms of intensity (mean) and shape (area) features. Fig. 7.7 displays two textural gray-level cooccurrence matrix (GLCM)-based features. According to the figure, the textural feature values cover $(\max(\text{Data}) - \min(\text{Data})) / \max(\text{Theory}) = (3.8 - 2.1) / 7 = 24.3\%$ (texture contrast) and $(0.067 - 0.025) / 1 = 4.2\%$ (second angular moment) of the possible feature value range.

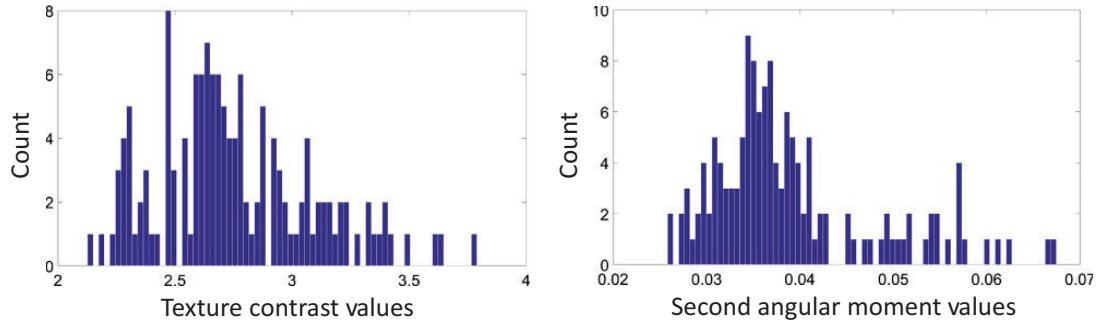


Fig. 7.7 Histograms of GLCM-based texture contrast (left) and texture second angular moment (right) features. Both were computed by using a Python library with parameters set to $\text{offset}=3$, $\text{angle}=0$ degrees, and the number of GLCM bins $=8$.

6. Variability of image features

We define the evaluation metrics for feature variability in [Section 6.1](#), present analyses of intensity, shape, and textural feature variability across five libraries in [Section 6.2](#), and discuss the sources of differences in [Sections 6.3 and 6.4](#).

6.1 Feature variability metric

For any two feature extractors containing the implementations of the same feature, we compute two numerical values per ROI. When the computation is applied to a set of ROIs, then it results in two vectors of numerical feature values, $\vec{V}_1 = (V_{1i})$ and $\vec{V}_2 = (V_{2i})$, where i is the index of the ROI. To compare the two vectors, we first calculate a vector of relative errors, $\vec{E}^{1,2} = (E_i^{1,2})$, that correspond to differences normalized by the average of V_{1i} and V_{2i} . Next, we count the elements of the relative error vector for which the error values exceed a specified threshold T in order to define the dissimilarity metric \mathbf{SF} :

$$E_i^{1,2} = |V_{1i} - V_{2i}| / (0.5 * (V_{1i} + V_{2i})) \quad (7.1)$$

$$\mathbf{SF} = \sum_{i=1}^n f_i \quad f_i = \begin{cases} 1 & \text{if } (E_i^{1,2} > T) \\ 0 & \text{otherwise} \end{cases} \quad (7.2)$$

where n is the number of ROIs, T is the user-defined error threshold, and the \mathbf{F} in \mathbf{SF} stands for “feature.” In our work, T is set to 0.01, which can be interpreted as a 1% deviation from the average. The purpose of T is to include only significant feature value differences. The error is normalized by the minimum value, which represents the worst-case error scenario. In this example, the dissimilarity metric (SF) is the number of colonies where the relative error was larger than 1%.

Table 7.3 Summary of a pair-wise comparison of feature extraction libraries (rows) in terms of the number of common (overlapping) features (CF) and the number of features with relative error larger than 1% (SF)

Feature extraction library pair	Intensity		Shape		Texture	
	SF	CF	SF	CF	SF	CF
Python versus CellProfiler	2	3	6	11	12	12
Python versus MaZda	0	1	0	3	12	12
Python versus ImageJ/Fiji	0	3	9	12	0	0
Python versus Java	0	1	0	11	0	0
CellProfiler versus MaZda	0	2	2	3	36	44
CellProfiler versus ImageJ/Fiji	0	5	8	8	0	0
CellProfiler versus Java	0	3	4	7	0	0
MaZda versus ImageJ/Fiji	2	4	3	3	0	0
MaZda versus Java	0	4	0	3	0	0
ImageJ/Fiji versus Java	1	6	2	10	0	0
Sum	5	32	34	71	60	68

6.2 Image feature variability analysis

Table 7.3 shows the results of image feature variability evaluation using the metric defined in Section 6.1. The rows report pairwise comparisons of feature extraction libraries. The columns are organized based on the three types of features (i.e., intensity, shape, and texture). The tabular entries document the values from the metric SF ($T=1\%$, $n=140$ cell colonies per feature) and the total number of common features in a pair of feature extraction libraries based on their definition.

For instance, out of 44 common textural features between MaZda and CellProfiler, 36 show a significant difference in values. Python and CellProfiler have 12 common textural features whose values show significant differences for all the features, which is similar to the evaluation results for Python and MaZda. For the five open-source feature extraction libraries (Python, CellProfiler, MaZda, ImageJ/Fiji, and Java), we found 5 out of 32 intensity features, 34 out of 71 shape features, and 60 out of 68 textural features to differ in their pairwise comparisons. These findings correspond to 15.6%, 47.9%, and 88.2% of intensity, shape, and textural features that differ in values, respectively.

6.3 Sources of image feature variations

To illustrate feature variability, we plotted the difference values between the feature value V_{ji} (j , feature extractor library index; i , ROI index) and the average of all feature values over all feature extractors containing the implementation of that feature. Figs. 7.8–7.10 provide such visualizations for one of the intensity, shape, and textural features, respectively. In Fig. 7.8, CellProfiler deviates from most feature extractors for mean intensity.



Fig. 7.8 Mean intensity feature (ID=24) differences over multiple regions of interest (ROIs) from cell colony images [20].

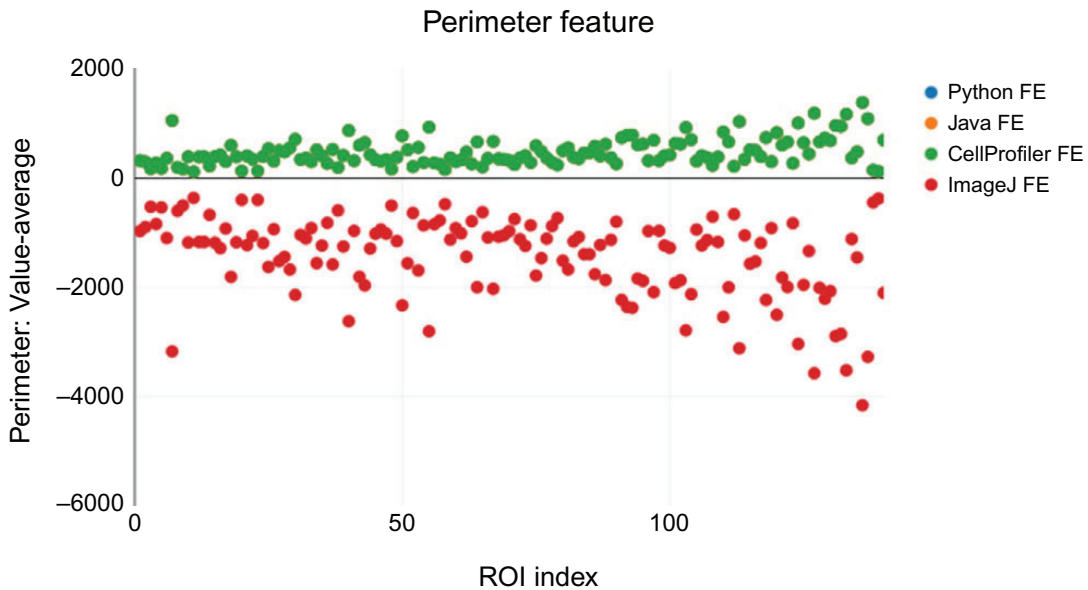


Fig. 7.9 Perimeter feature (ID=8) differences over multiple regions of interest (ROIs) from cell colony images [20]. The unit is image pixels.

In Fig. 7.9, ImageJ/Fiji reports different values from the rest of feature extractors for the perimeter (shape feature). Fig. 7.10 shows gray-level cooccurrence matrix (GLCM)-based texture contrast difference values for the MaZda, Python, and CellProfiler libraries. In this illustration, CellProfiler and MaZda agree in values but disagree with the values from Python.

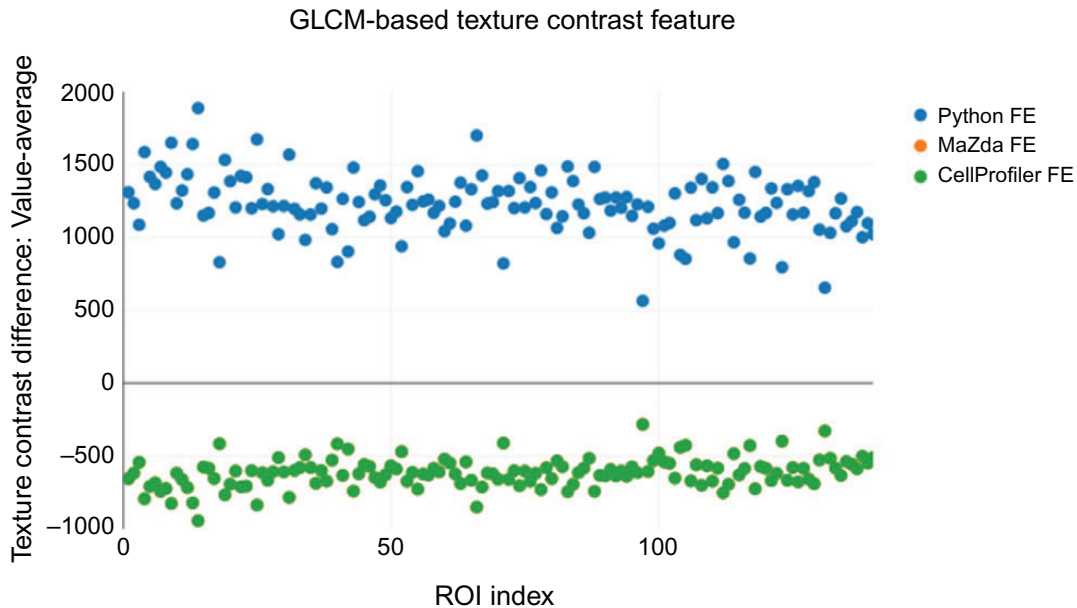


Fig. 7.10 Texture contrast difference feature (ID=28) over 140 ROIs. The values are plotted as the difference between the computed feature and the average of all three values from MaZda, CellProfiler, and Python.

Next, we have analyzed some of the sources of variation for the features referred to in the [Appendix](#). The sources of feature variations included theoretical formulas for the same image feature, the physical units used to represent pixel-based measurements, the definitions of objects in images (called regions of interest or ROIs), algorithmic implementations, and the number of exposed parameters to a user in multiple feature extractors. Our analysis also includes features requiring special attention because they are prone to variation.

6.3.1 Intensity features

Kurtosis and Skewness: The kurtosis disagreement in values between software packages depends on whether the excess kurtosis or kurtosis is implemented (fixed, offset by 3). Similarly, one has to be aware of multiple definitions of skewness (e.g., sample versus population skewness).

Histogram bins for intensities represented by more than 8 bits per pixel (special attention): ImageJ/Fiji uses the max value +1 as the upper value of the last bin. It assumes that the lower value of the first bin is always zero.

Python and its numpy library provide two definitions: $B = \text{histogram}(X, N)$ uses N equally spaced bins within the appropriate range for the given image data type. The returned image B has no more than N discrete levels. $B = \text{histogram}(X, \text{edges})$ sorts X into bins with the bin edges specified by the vector, edges . Each bin includes the left edge but does not include the right one. The last bin is an exception because it includes both edges.

6.3.2 Shape features

Perimeter and Circularity: The perimeter variability comes from the fact that algorithmic implementations differ in counting interior or exterior pixels, use 4 or 8 connectivity of pixels, and might interpolate between the boundary points. Circularity is inversely proportional to the perimeter squared.

Solidity: The same definition of solidity is used by Python and ImageJ/Fiji (area/convex area). The difference between these values comes from the convex area differences because the implementations vary.

Centroid (and Bounding Box): The centroid and the bounding box are both subject to the choice of the reference coordinate system (+col \sim x; +row \sim y or +row \sim -y). In addition, the bounding box of a ROI is defined by its upper-left-corner coordinate and its width and height. However, the bounding coordinates might vary depending on the choice of values as integers or floats in a raster image.

Euler number (special attention): The Euler number definition is the number of objects (ROIs) minus the number of holes. The value might differ depending on the assumptions about the number of ROIs (Python assumes #ROIs = 1).

Orientation (special attention): The orientation is the angle between the major axis of a given ROI and the x -axis. It can be computed using two mathematical formulas: (1) $\theta = \text{atan}\left(\frac{V_y}{V_x}\right)$, where atan is the arctangent function and V_x and V_y are the x - and y -decompositions of the major axis of the ROI; and (2) $\theta = \frac{1}{2}\text{atan2}\left(\frac{2I_{xy}}{I_{xx}-I_{yy}}\right)$, where I_{xx} and I_{yy} are the second moment of area along the x - and y -axes and I_{xy} is the product moment of area. These two formulas are equivalent if the first one is computed in the range of $[-\pi/2, \pi/2]$ using atan , and the second one in the range of $[-\pi, \pi]$ using atan2 . The variations are observed if different value ranges or angular units would be reported by selected software packages. The range can be either $[-\pi/2, \pi/2]$ or $[-\pi, \pi]$, and the unit can be expressed either in radians or degrees. In addition, the sign of the output angle depends on the coordinate system (image coordinate or graph coordinate system with clockwise or counterclockwise axes).

6.3.3 Textural Features

In comparison to intensity and shape feature variability evaluations, the evaluation of textural features is more challenging due to a large space of possible multistep computations. The reason for this lies in the difficulty of describing image texture. According to Ref. [38], textures are complex visual patterns composed of entities that have characteristic brightness, color, slope, size, and other traits. To capture the textural complexity, we decomposed the process of extracting textural features into four algorithmic steps:

- (1) Pixel selection based on regions of interest (ROIs)
- (2) Normalization of intensities

Table 7.4 Parameters associated with each textural feature extraction step

Algorithmic steps	Parameters
Selection of pixels	ROI image mask
Normalization	Type (none, min-max, mean \pm standard deviation)
Transformation	Spatial scale, spectral bins
Computation of statistics	Mathematical formula

(3) Spatospectral transformation of pixels

(4) Computation of statistics from transformed pixels

Table 7.4 provides a high-level summary of all parameters associated with each textural feature extraction step.

Because objects of interest can take arbitrary shapes, selection of pixels via a ROI mask allows us to extract textural features over only the ROI area. The purpose of normalization is to adjust for various image-acquisition settings (e.g., dynamic ranges of images) to enable the comparison of extracted textural features. The goal of spatospectral image transformations is to capture the structure of texture in a new coordinate space, such as by applying Fourier, Gabor, wavelet, run length coding (RLC), or gray-level cooccurrence matrix (GLCM) transforms. Finally, the objective of computations of statistics is to reduce the dimensionality of the resulting textural feature.

To narrow down the number of textural features and their parameters, we chose GLCM [39] as the transformation of interest because it has been implemented in many software libraries and is used frequently in biological publications [8, 40]. For the GLCM transform, spatial scale is defined by offset and angle parameters. Spectral bins map the acquired dynamic range in bits per pixels (BPP) to a smaller range and define the GLCM size, denoted by N (i.e., N corresponds to the number of distinct gray levels). In addition to defining the spectral bins, the cooccurrence in GLCM can be computed by considering black-white and white-black transitions as the same (both transition counts are added and the GLCM matrix is symmetric) or different (each transition is unique and the GLCM matrix is asymmetric). Finally, when computing statistics from GLCMs, the indexing of bins can vary between 0 and $(N-1)$ or between 1 and N , which affects numerical values derived using GLCM indices. This can be seen from the equations for the texture difference average:

$$F^{(1)} = \sum_{i=0}^{N-1} i p_{x-y}(i) \quad (7.3)$$

$$F^{(2)} = \sum_{i=1}^N i p_{x-y}(i), \quad (7.4)$$

where i is the index of the GLCM matrix, $p(i, j)$ are the entries in the GLCM, and

$$p_{x-y}(i) = \sum_{|x-y|=i} p(i, j). \quad (7.5)$$

The challenge of setting up the algorithmic parameters of multiple libraries lies in the lack of access to some of these parameters (and their supported ranges). Thus, a user is left with the option of only approximating parameter settings. We summarized the key parameter differences among the Python, CellProfiler, and MaZda libraries in [Table 7.5](#). With the limited access to parameters listed in this table, we approximated the selection of pixels by zeroing pixels outside of ROIs and left others intact. All other parameters were set to the same values.

A detailed list of GLCM-based features is provided in the [Appendix](#). It is important to note that in order to extract statistically robust GLCM-based feature values, hundreds of pixels in ROI are needed.

These complex sequences of computations and their algorithmic parameters needed for extracting GLCM-based textural features become the sources of textural feature variability. The variability originates not only from the inconsistent settings of many parameters, but also from the fact that they are inaccessible. After a careful inspection of the software, we identified additional sources of variabilities, such as naming discrepancies (e.g., GLCM angular second moment versus GLCM energy), limited support of image data types (e.g., larger than 8 BPP), and limited ranges of input parameters (e.g., GLCM angles constrained to 0, 45, 90, and 135 degrees).

6.4 Discussion

The evaluation of feature variability across image feature extraction libraries depends on the chosen metric and its parameters. The metric used in our analysis is based on a relative error and depends on a user-specified threshold that was set empirically to 1% of the relative error. Feature variation evaluations also depend on a unit of feature measurement. Our analysis uses pixel units. Note that ImageJ/Fiji reports all measurements in physical units (i.e., μm), and therefore, the ImageJ/Fiji settings must be modified. In our study, we set the units in ImageJ/Fiji to be in pixels.

To compare the magnitude of feature variability across intensity, shape, and textural feature types, we plotted the evaluation results in [Fig. 7.11](#) for 8 intensity features,

Table 7.5 Key parameter differences in GLCM-based features extracted by Python, CellProfiler, and MaZda

Library	ROI Support	Norm. Type	Transformation		Statistics GLCM indexing
			GLCM size N	Sym.	
Python	No	Min-max	256×256	A	$0 - (N - 1)$
CellProfiler	Yes	Min-max	8×8	A	$0 - (N - 1)$
MaZda	Yes	None	8×8	S	$1 - N$

Sym. is for symmetry, A is asymmetric, and S is symmetric. Norm. stands for normalization.

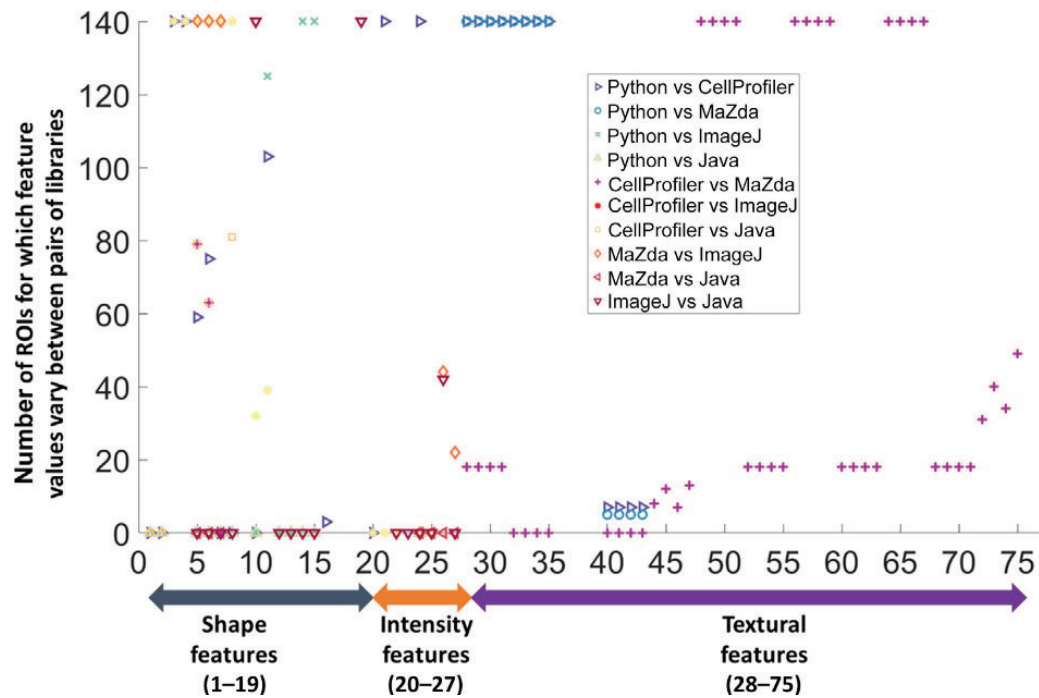


Fig. 7.11 Image feature variability based on the metric SF , evaluated over three pairs of feature extraction libraries. Feature categories are indicated on the x -axis and index-to-feature name mapping is provided in the [Appendix](#).

19 shape features, and 48 textural features that are unique and common between pairs of feature extraction libraries. The image shows a total of 75 unique features on the x -axis (the mapping of a feature index to its feature name is in the [Appendix](#)) and the metric SF on the y -axis [the count of ROIs (cell colonies) for which a given feature value varies more than 1% between two feature extraction libraries (see the inset in [Fig. 7.11](#))].

7. Feature-based classification

In order to evaluate the effect of image feature variability on feature-based classification, we used the manually assigned labels (homogeneous, heterogeneous, and dark) in 140 colonies. The labels are assigned based on the distribution of the Oct4 marker in each colony. From images of the Oct4 marker, we extracted 170 textural features per cell colony using Python, CellProfiler, and MaZda libraries. Out of 170 features, we identified 68 common features between pairs of feature extractors. For these 68 features and the colony labels, we built a classification model to assign each colony to one of the categories based on its feature values. The classification outcomes were evaluated based on the classification variability metric, presented in the next section.

7.1 Classification variability metric

Given the i th ROI (cell colony) in the set of ROIs and two classification labels, L_i^1 and L_i^2 , obtained from feature-based classifications of the i th ROI with the same input features provided by two feature extraction libraries, the metric reports the number of ROIs for which $L_i^1 \neq L_i^2$. The metric does not measure the actual classification accuracy, but rather the difference in classification outcomes across pairs of feature extraction libraries. The **SC** metric definition is presented here (**C** in **SC** stands for classification):

$$SC = \sum_{i=1}^n f_i \quad f_i = \begin{cases} 1 & \text{if } (L_i^1 \neq L_i^2) \\ 0 & \text{otherwise} \end{cases} \quad (7.6)$$

7.2 Classification variability analysis

To build a classification model, we randomly selected half of the 140 manually labeled cell colonies for training and the remainder for testing. The training and testing steps are performed for one feature at a time. The random split of 140 cell colonies is performed 10 times, and the majority label is selected as the final prediction for each colony.

Table 7.6 summarizes the number of features leading to feature-based classification variability and the total number of common textural features for each pair of feature extractors (rows). For this analysis, we chose the binary classification tree because it does not include any randomness during its execution. Note that the classifier choice is not important because we are not analyzing the accuracy of the results, but rather the difference in the classification outcomes. If all classification labels were the same regardless of feature extraction libraries, then SC would be zero and nonzero SF would have zero impact on SC .

7.3 Discussion

To compare the magnitude of classification variability across intensity, shape, and textural feature types, we plotted the evaluation results as illustrated in Fig. 7.12. It shows a total of 75 unique features on the x -axis (the mapping of a feature index to its feature name is in

Table 7.6 Summary of pair-wise comparison of feature extraction libraries (rows) in terms of their common features (CF) by definition and those common features that yield different classification labels according to the classification variability metric SC

Feature extraction library pair	Intensity		Shape		Texture	
	SC	CF	SC	CF	SC	CF
Python versus CellProfiler	0	3	7	11	12	12
Python versus MaZda	0	1	0	3	12	12
CellProfiler versus MaZda	0	2	2	3	42	44
Sum	0	6	9	17	66	68

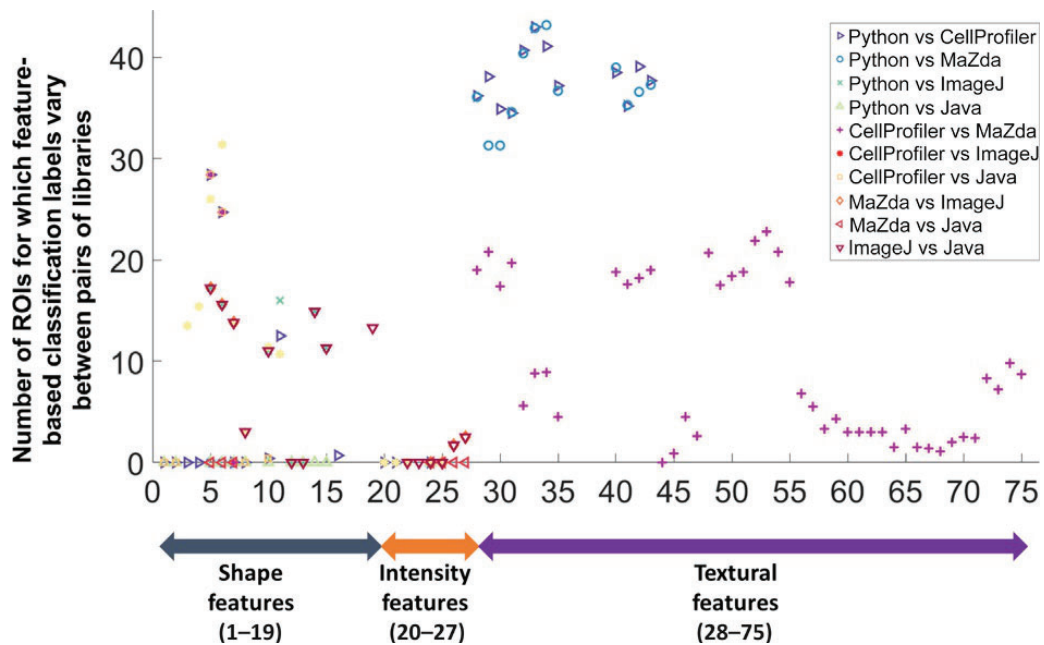


Fig. 7.12 Feature-based classification variability based on the metric SC evaluated over three pairs of feature extraction libraries. Feature categories are indicated on the x-axis, and index-to-feature name mapping is provided in the [Appendix](#).

the [Appendix](#)) and the metric SC on the y-axis [the count of ROIs (cell colonies) for which classification labels differ between two feature extraction libraries (see the inset in [Fig. 7.12](#))].

Based on this analysis, we concluded that the variation of intensity features had no impact on classification. However, 9 out of 17 shape features (52.9%) and 66 out of 68 textural features (97.1%) had an impact on classification outcomes.

8. Summary

This chapter focused on object measurements or features from 2D microscopy images. In the presence of many open-source and commercial image feature extraction libraries, object measurements from images were evaluated in terms of their variability across multiple open-source libraries. The impact of object feature variability was quantified by analyzing feature-based classification outcomes. By characterizing these feature variations across Python scikit-image, CellProfiler, MaZda, ImageJ/Fiji, and in-house Java libraries, we concluded 15.6% of 32 intensity features, 47.9% of 71 shape features, and 88.2% of 68 textural features differ in value. These feature variations had no impact on classification outcomes based on intensity features but had negative impact on classification based on

shape features in 52.9% and based on textural features in 97.1% of single feature classifications.

As one part of this work, we are also disseminating the web image processing pipeline (WIPP) system, which integrates heterogeneous image feature libraries, executes feature calculations, and shares hyperlinked image feature values with computational provenance artifacts. Scientists can use WIPP for publishing traceable 2D image measurements. The WIPP system installation is simplified using a Docker container and is available at <https://isg.nist.gov/deepzoomweb/software/wipp>. We plan to upgrade the integrated image extraction libraries to their latest versions over time.

The study focused exclusively on open-source image feature extraction libraries because otherwise, the sources of feature variability could not be identified. We did not include software that uses commercial platforms, as explained in the disclaimer at the end of this chapter. While feature engineering inside deep-learning models has demonstrated a good deal of promise in recent years, we did not include features from convolutional layers of deep-learning models in this study because the mathematical and semantic meanings of those features are missing. Thus, relating those features would be almost impossible across feature extraction libraries, deep-learning architectures, parameters, and selections of training data subsets. Finally, we did not include hardware-accelerated [central processing unit (CPU)/graphics processing unit (GPU) or field-programmable gate array (FPGA)] software for image feature extraction because GPU-accelerated image feature extraction has been missing in the open-source libraries.

The overall study documented the need for standard definitions of widely used image features. In the absence of standards, there is a need for gathering provenance information to achieve transparency of image measurements and reproducible research. As of now, if a laboratory performs a complex experimental design for drug treatment or scientific discoveries, then feature-based classification results of data analyses will depend on the choice of the image feature extraction library. Thus, scientists must keep provenance information about the libraries and their parameters used for their analysis. Otherwise, the results would not be reproducible between laboratories, and the ultimate conclusions derived from the same experimental data may differ.

Standard definitions of widely used image features would also make object measurements more likely to be optimized for speed of execution and memory consumption. As reported in Ref. [37], the in-house Java package has the fastest CPU computation time for a large number of small images (8162 ROIs in 238 images of size 446 kB) with the speed-up factor of 2.81 (Python) and 36.55 (ImageJ/Fiji). However, against a large image (200 ROIs in one image with a size of 1.9 GB), Python is the most efficient, with a factor of 3.92 (ImageJ/Fiji) and 2.51 (Java), while CellProfiler would not run due to its memory consumption exceeding our 10-GB limit. In the future, these benchmarks could motivate the development of memory-efficient and hardware-accelerated implementations.

Appendix: Online information about the work

The reader can find detailed information about the work discussed in this chapter and download the metadata files from <https://isg.nist.gov/deepzoomweb/resources/featureVariability/index.html>.

The web pages contain information about the feature list, the feature index to feature name mapping, and all the numerical data required to plot Figs. 7.8–7.12.

The numerical feature values have been postprocessed to deliver interactive interfaces to traceable results of this image feature variability study. The set of interactive graphs and tables is publicly available from the abovementioned web page. The web pages cited in this chapter allow readers to identify interactively image features that differ across open-source feature extraction libraries and to trace the numerical feature values to the original (persistent) images used for quantification.

Acknowledgments

We would like to acknowledge Mohamed Ouladi, Antonio Cardone, Antoine Vandecreme, Julien Amelot, Philippe Dessauw, and Antoine Gerardin from the Computational Science in Metrology project at NIST, who have contributed to the code development of the web image feature extraction system. We also would like to acknowledge the following NIST colleagues for their support and input: Kiran Bhadriraju, Michael Halter, Subhash Sista, John Elliott, and Anne Plant (of the Material Measurement Laboratory at NIST) for acquiring the test image collection and creating the cell colony annotations.

Disclaimer

Commercial products are identified in this document in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the products identified are necessarily the best ones available for this purpose.

We used the words *Python* and *Java* to refer to specific implementations of image feature extractors (the Python scikit-image and in-house developed Java libraries) written in these distinct programming languages. While we used the two words as abbreviated names for libraries, we do not intend to imply that one of the programming languages is better than the other.

References

- [1] S. van der Walt, et al., scikit-image: image processing in Python, *PeerJ* 2 (2014) e453.
- [2] M. Lamprecht, D. Sabatini, A. Carpenter, CellProfilerTM: free, versatile software for automated biological image analysis, *BioTechniques* 42 (1) (2007) 71–75.
- [3] P.M. Szczypiński, M. Strzelecki, A. Materka, A. Klepaczko, MaZda—A software package for image texture analysis, *Comput. Methods Prog. Biomed.* 94 (1) (2009) 66–76.
- [4] J. Schindelin, et al., Fiji: an open-source platform for biological-image analysis, *Nat. Methods* 9 (7) (2012) 676–682.
- [5] N.V. Orlov, L. Shamir, T. Macura, J. Johnston, D.M. Eckley, I.G. Goldberg, WND-CHARM: Multi-purpose image classification using compound image transforms, *Pattern Recogn. Lett.* 29 (11) (2008) 1684–1693.

- [6] Open source development, OpenCV (Open Source Computer Vision Library), *web page*, (2000). [Online]. Available: <https://opencv.org/>. (Accessed 19 December 2017).
- [7] Oracle Java 1.7, Java ImageIO, *web page*(1993) [Online]. Available: <https://docs.oracle.com/javase/7/docs/api/javax/imageio/ImageIO.html>. (Accessed 19 December 2017).
- [8] K. Bhadriraju, et al., Large-scale time-lapse microscopy of Oct4 expression in human embryonic stem cell colonies, *Stem Cell Res.* 17 (1) (2016) 122–129.
- [9] P. Bajcsy, et al., Enabling Interactive Measurements from Large Coverage Microscopy, *IEEE Comput.* 49 (7) (2016) 70–79.
- [10] A. Vandecreme, et al., From image tiles to web-based interactive measurements in one stop, *Micros. Today* 25 (1) (2017) 18–27.
- [11] M.V. Boland, R.F. Murphy, A neural network classifier capable of recognizing the patterns of all major subcellular structures in fluorescence microscope images of HeLa cells, *Bioinformatics* 17 (12) (2001) 1213–1223.
- [12] A.E. Carpenter, Extracting rich information from images, *Methods Mol. Biol.* 486 (2009) 193–211.
- [13] The Mathworks Inc., MATLAB and Image Processing Toolbox Release 2015b, *MathWorks Inc.*, 2015. [Online]. Available: <http://www.mathworks.com/help/images/index.html>. Accessed 16 March 2016.
- [14] G. David Lowe, Distinctive image features from scale-invariant keypoints, *Int. J. Comput. Vis.* 60 (2) (2004) 91–110.
- [15] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool, Speeded-Up Robust Features (SURF), *Comput. Vis. Image Underst.* 110 (3) (2008) 346–359.
- [16] N. Dalal, W. Triggs, Histograms of Oriented Gradients for Human Detection, in: *2005 IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. CVPR05*, vol. 1, no. 3, 2004, pp. 886–893.
- [17] J. Kannala, E. Rahtu, BSIF: Binarized statistical image features, in: *21st Int. Conf. Pattern Recognit.*, no. Icp, 2012, pp. 1363–1366.
- [18] B. Yang, S. Chen, A comparative study on local binary pattern (LBP) based face recognition: LBP histogram versus LBP image, *Neurocomputing* 120 (2013) 365–379.
- [19] J. Ren, X. Jiang, J. Yuan, Relaxed Local Ternary Pattern for Face Recognition, in: *ICIP 2013*, no. 1, 2013, pp. 3680–3684.
- [20] B. Julesz, A brief outline of the texton theory of human vision, *Trends Neurosci.* 7 (2) (1984) 41–45.
- [21] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, F.E. Alsaadi, A survey of deep neural network architectures and their applications, *Neurocomputing* 234 (2017) 11–26.
- [22] S. Bahrapour, N. Ramakrishnan, L. Schott, M. Shah, Comparative Study of Deep Learning Software Frameworks, *Arxiv - a Repos. Electron. Prepr*, 2016.
- [23] S. Mallat, Understanding Deep Convolutional Networks, *Philos. Trans. A* 374 (20150203) (2016) 1–17.
- [24] V. Badrinarayanan, A. Kendall, R. Cipolla, SegNet: A Deep Convolutional Encoder–Decoder Architecture for Image Segmentation, *Arxiv—a Repos. Electron. Prepr*, 2015, pp. 1–14.
- [25] J. Heaton, An Empirical Analysis of Feature Engineering for Predictive Modeling, *Arxiv - a Repos. Electron. Prepr*, 1701.07852, 2017, p. 6.
- [26] H.-F. Yu, et al., Feature engineering and classifier ensemble for KDD cup, in: *JMLR: Workshop and Conference Proceedings*, 2010, 2010, pp. 379–388.
- [27] P. Szczypinski, M. Strzelecki, A. Materka, A. Klepaczko, Mazda software; a computer program for calculation of texture parameters (features) in digitized images, *Comput. Methods Prog. Biomed.* 94 (1) (2009) 66–76.
- [28] M. Strzelecki, P. Szczypinski, A. Materka, A. Klepaczko, A software tool for automatic classification and segmentation of 2D/3D medical images, *Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrom. Detect. Assoc. Equip.* 702 (Feb. 2013) 137–140.
- [29] A. Materka, *MaZda User’s Manual*, Lodz, 1999.
- [30] I. NIST, Image Features, *web page*. [Online]. Available: <https://isg.nist.gov/deepzoomweb/stemcellfeatures>, 2016. Accessed 19 December 2017.
- [31] E. Deelman, et al., Pegasus a workflow management system for science automation, *Futur. Gener. Comput. Syst.* 46 (2015) 17–35.

- [32] M.N. Gurcan, L. Boucheron, A. Can, A. Madabhushi, N. Rajpoot, B. Yener, Histopathological image analysis: a review, *IEEE Rev. Biomed. Eng.* 1 (2009) 147–171.
- [33] J. Chalfoun, M. Majurski, K. Bhadriraju, S. Lund, P. Bajcsy, M. Brady, Background intensity correction for terabyte-sized time-lapse images, *J. Microsc.* 257 (3) (2015) 226–238.
- [34] J. Chalfoun, M. Majurski, T. Blattner, W. Keyrouz, P. Bajcsy, M.C. Brady, MIST: Accurate and scalable microscopy image stitching method with stage modeling and error minimization, *Nat. Sci. Rep.* (2017).
- [35] J. Chalfoun, M. Majurski, A. Peskin, C. Breen, P. Bajcsy, Empirical gradient threshold technique for automated segmentation across image modalities and cell lines, *J. Microsc.* 260 (1) (2014) 86–99.
- [36] J. Chalfoun, M. Majurski, A. Dima, M. Halter, K. Bhadriraju, M. Brady, Lineage mapper: a versatile cell and particle tracker, *Sci. Rep.* 6 (2016) 36984.
- [37] M. Simon, J. Chalfoun, M. Brady, P. Bajcsy, Do We Trust Image Measurements? Variability, Accuracy and Traceability of Image Features, in: *IEEE International Conference on Big Data*, 2016.
- [38] A. Materka, M. Strzelecki, *Texture Analysis Methods – A Review*, Lodz, 1998.
- [39] R.M. Haralick, K. Shanmugam, I. Dinstein, Textural Features for Image Classification, *IEEE Trans. Syst. Man. Cybern.* 3 (6) (1973) 610–621.
- [40] T. Kobayashi, D. Sundaram, K. Nakata, H. Tsurui, Gray-level co-occurrence matrix analysis of several cell types in mouse brain using resolution-enhanced photothermal microscopy, *J. Biomed. Opt.* 22 (3) (2017), 36011.