Do We Trust Image Measurements?

Variability, Accuracy and Traceability of Image Features

Mylene Simon, Joe Chalfoun, Mary Brady, and Peter Bajcsy
Information Technology Laboratory
National Institute of Standards and Technology
100 Bureau Drive, Gaithersburg, MD 20899
{mylene.simon, joe.chalfoun, mary.brady, peter.bajcsy}@nist.gov

***Abstract*—The paper addresses the problem of understanding quality of image measurements extracted using widely used software libraries from large images. Image measurements (features) are extracted using software packages that vary in terms of programming languages, theoretical formulas for the same image feature, algorithmic implementations, input parameters, units of measurements, and definitions of image regions of interest. Our motivation is to quantify numerical variability of image features across software packages and determine image accuracy with respect to reference images. In addition, our objective is to enable scientists to extract any image feature of interest from heterogeneous software libraries and gain provenance of every extracted numerical feature value.**

We pursue this objective by designing a client-server system that integrates image feature extractions from open source libraries such as ImageJ/Fiji, Python (scikit-image), CellProfiler, WND-CHARM, and in-house Java software packages. The system becomes useful for evaluating quality of image measurements, leveraging distributed computational resources for feature computations over big image data, sharing resulting feature values, and reproducing the feature values based on provenance. As an application of the designed system, we report the quality evaluations of 319 image features extracted using ImageJ/Fiji, Python (scikit-image), CellProfiler and in-house Java software packages with 43 duplicate features across the four packages. Using the normalized difference as metric, we identified 6 out of the 43 common features to differ over 1% in value and discuss the sources of these numerical differences.

Keywords-Big Data Science and Foundations: Data and Information Quality for Big Data

I. INTRODUCTION

Quantitative imaging and image informatics depend on taking image measurements over a region of interest. These image measurements are frequently denoted as image or object features. They are extracted by applying a wide variety of mathematical operations to image pixel values implemented in software. The software implementations vary in terms of programming languages, theoretical formulas for the same image feature, algorithmic implementations, input parameters, units of measurements, and definitions of image regions of interest. The motivation of our work is (1) to understand the variability of image feature extractions and (2) to enable scientists to compute any feature of interest from widely-used heterogeneous software libraries and gain provenance of every extracted numerical feature value. The word provenance refers to the record of image feature value's ultimate computational derivation and passage through various software tools.

To study the variability, the challenges lie in (a) identifying the same image features in multiple software packages, (b) setting their input parameters consistently, and (c) establishing pairs of reference images and image features for evaluations. In addition to the knowledge about variability, one would like to gather and access information about each feature extraction so that the feature values are traceable, reproducible, and executable in parallel on big image data. The challenges of enabling extraction of traceable image features lie in (a) integrating heterogeneous software, (b) gathering and hyperlinking all provenance information about feature extraction, and (c) designing a client-server system that enables data upload, configuration, computationally scalable feature computation, and access to feature values and all provenance artifacts to re-execute the image measurements. All the above challenges are mapped into two basic questions and associated problems. (1) What image features are trustworthy across software packages? (2) What practical solution would improve our trust in image features?

We approach the first problem by considering 319 image features extracted using ImageJ/Fiji, Python (scikit-image), CellProfiler and in-house Java software packages with 43 duplicate features across the four packages. Using the normalized difference metric, we identified 6 out of 43 features to differ over 1 % in value. We analyzed the sources of these numerical differences for some features to raise the awareness of community users. We approach the second problem by designing a web system with (1) interfaces to loading images and extracting image features while utilizing distributed computational resources, (2) access to feature implementations in several software packages, (3) a provenance information gathering mechanism, and (4) feature values hyperlinked with all computational provenance artifacts.

While there is an abundance of image feature implementations, the quality of feature values in terms of accuracy, variation, and execution traceability has not been evaluated. Many image features have been implemented in academic environments [1], [2], [3], commercial platforms [4], or in publicly available image libraries [5], [6], [7]. The use of these image features is primarily for classification (find the most discriminative or predictive features, given a certain number of classes) and for discovery (understand statistical and semantic image characteristics based on image features). In the context of discovery, our focus is on the quality of image feature with respect to a semantically meaningful object rather than low-level image descriptors (e.g., Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF), Histograms of Oriented Gradients (HOG), Local Phase Quantization (LPQ), Binarized Statistical Image Features (BSIF), Local Binary Pattern (LBP) or Local ternary patterns (LTP)). Thus, our analysis is about accuracy of those image features that describe intensity, shape, or texture properties of an object. Regarding computational scalability and traceability of image feature execution, we are leveraging work in the area of scientific workflows, such as the capabilities of Pegasus [8]. In comparison to many existing scientific workflows, our work integrates heterogeneous software using a common file interface as opposed to wrapping software to a pre-defined workflow programming interface. Furthermore, our work is filling the gap in traceability of image feature execution by designing a community resource for sharing and reproducing scientific measurements.

The novelties of the work are (a) in documenting image feature variability across four software packages and (b) in designing a software plug-and-play framework for adding image feature extraction plugins, conducting image feature comparisons, and for delivering image feature values hyperlinked with computational provenance information.

The paper is organized as follows. Sections II, III, and IV are devoted to variability, accuracy and traceability of image features respectively. Sections II and III consist of evaluation setup, numerical evaluation, and deeper analysis. Section IV focuses on design and capabilities needed to extract any image feature of interest from heterogeneous software libraries and gain provenance of every extracted numerical feature value. An overall summary is provided in Section V.

II. Variability of Image Features

The variability study provides numerical evidence about the differences in various implementations of the same image features.

A. *Feature Extraction Software*

We evaluated four software packages with the total of 218 unique features. The subsets of unique features are implemented in Python (40 features), ImageJ/Fiji (33 features), Java (74 features), and CellProfiler (101 features). Python features were implemented on top of an existing image processing library (scikit-image [5]), ImageJ/Fiji [6] features were implemented as a plugin using the ImageJ application programming interface (API), and Java features were implemented from scratch at NIST [9].

We focused primarily on intensity and shape features in this work.

B. *Test Images*

We chose the live phase contrast 3T3 images comprised of 238 images and a total of 8162 cells with different shapes and sizes (Figure 1) [10] to analyze common feature value variability between all 4 software packages. The cells are segmented using the EGT technique [11], and the masks are saved as labeled images. The features are computed on top of the segmented Regions of Interests (ROIs).

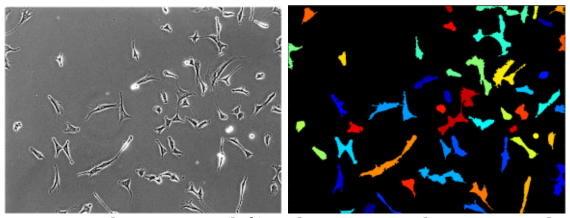


Figure 1: Example test image (left) and its corresponding segmented mask (right). Each ROI in the segmented mask has a unique randomly chosen color for display purposes.

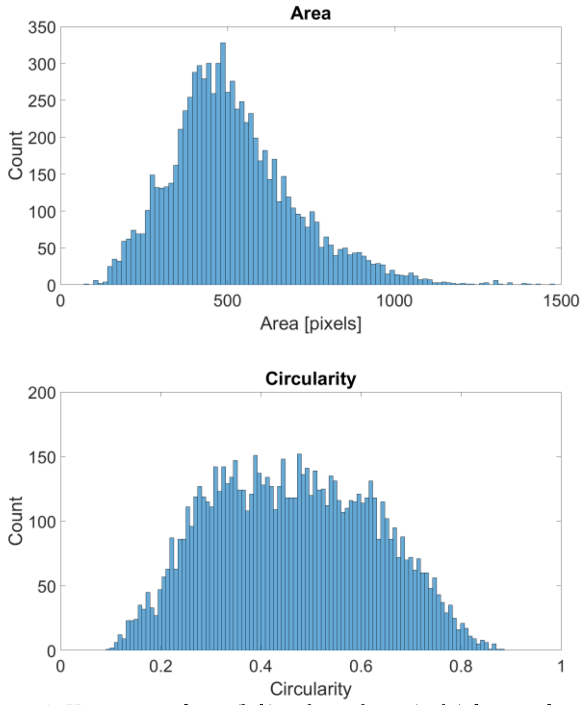


Figure 2: Histograms of area (left) and circularity (right) features from the objects defined by test images and their masks.

The measured images were selected for testing over synthetic images because (a) we did not have mathematical models for generating synthetic images that span a wide range of each image feature, and (b) we found the measured images to be a good initial approximation of the value range of each image feature. Figure 2 shows the histogram of test image measurements for area and circularity features.

C. *Evaluation Metric*

Given two vectors of feature values V_1 and V_2 computed over a set of ROIs (image segments) by two software implementations of the same feature, we compute their dissimilarity metric S as the sum of relative errors E_i^m normalized with respect to the minimum of the two values from the vectors V_1 and V_2 that exceed a given threshold:

$$S = \sum_i (E_i^m > T) \quad (1)$$
$$E_i^m = |\mathbf{V}_{1i} - \mathbf{V}_{2i}| / \min(\mathbf{V}_{1i}, \mathbf{V}_{2i})$$

The index i = 1,…,n and n is the number of ROIs. T is the user defined error threshold defined as 1 % of E_i^m in our work. The purpose of T is to detect substantial feature differences. The error is normalized by the minimum value which conveys the worst case error scenario.

D. *Image Feature Variability Analysis*

TABLE I. shows the results of feature variability evaluations using the aforementioned metric. The “Agree” column indicates when software have less than 1 % error across all 8162 test cells. The “Disagree” column indicates whether there is an error larger than 1 % across 8162 test cells between the tools that agree and the ones that disagree. The “Absent” column is used to denote with tools do not have an implementation of a given feature.

TABLE I. SUMMARY OF COMMON FEATURE VARIABILITY BETWEEN TOOLS BASED ON METRIC S (I = IMAGEJ, J = JAVA, P = PYTHON, C = CELLPROFILER).

Feature name	Agree	Disagree	Irrelevant
1- Perimeter		P,I,J,C	
2- Solidity	P, C	I	J
3- Circularity		I, J	C, P
4- Kurtosis		I, J	C, P
5- Bounding_Box_X	P, J	I	C
6- Bounding_Box_Y	P, J	I	C

Figure 3 illustrates the perimeter differences D_{ji} between its feature value V_{ji} and the average m_i of all three computed perimeter values per region of interest (i.e., cell segment). The feature difference follows the formula below:

$$D_{ji} = (V_{ji} - m_i) \quad (2)$$
$$m_i = \frac{1}{3}\sum_{j=1}^{3} V_{ji}$$

The index i = 1,…,n; j = 1, 2, 3, j is the software index, and n is the number of ROIs, The perimeter values range between 44.4 and 542.5 pixels in the set of randomly chosen 64 ROIs from the 8162 available ones. The number 64 corresponds to approximately the number of ROI present in two figures. This random data sampling is done only for visualization purposes.

E. *Sources of Feature Variations*

We have summarized our analysis of some of the sources of feature variations for the features listed in TABLE I. The summary also includes features requiring “special attention” since they are prone to variations.

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 perimeter squared.

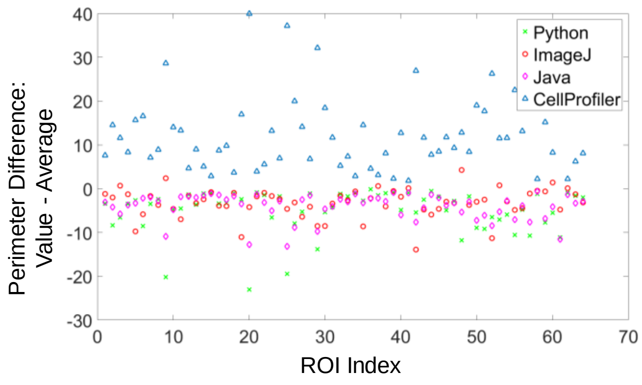


Figure 3: Perimeter feature differences over multiple regions of interests (ROIs). The unit is image pixel.

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

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

Centroid (and Bounding Box): The centroid and the bounding box are both subject to the choice of the reference coordinate system (+col ~ x; +row ~ y or +row~ -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 be vary depending on the choice of values as integers or floats in a pixelated image.

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

Histogram bins for intensities represented by more than 8 bits per pixel (special attention): ImageJ uses the max

value plus one 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 provides two definitions. B = 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 = histogram(X,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 edge. The last bin is an exception since it includes both edges.

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) $\theta = \text{atan}(\frac{V_y}{V_x})$ 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; (2) $\theta = \frac{1}{2}\,atan2\left(\frac{2I_{xy}}{I_{xx}-I_{yy}}\right)$ 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. Range can be either $[-\pi/2, \pi/2]$ or $[-\pi, \pi]$ and the unit can be either radian or degree. In addition, the sign of the output angle depends on the coordinate system (image coordinate or graph coordinate system with clockwise or counter clockwise axes).

F. Discussion

Absolute value of feature variability depends on the choice of a metric. The current metric has a user-specified threshold that was set empirically to 1 % of the relative error E_i^m in our analysis. In addition, the current analyses do not include texture features because we encountered a large variety of definitions, naming inconsistencies, and hard-coded parameters. The work on including texture features is in progress. We would also note that the feature comparison is conducted using pixel units. Among the evaluated software packages, ImageJ reports all measurements in physical units (i.e., micrometers) that a user should be aware of.

III. Accuracy of Image Feature Implementations

Accuracy analysis is based on two key components: (1) generation of synthetic images and their corresponding reference feature values, and (2) a metric to compute the error between reference and computed values. In this study, we used a theoretical feature value as the reference and assumed that the generated synthetic images are very close representations of analog shapes associated with the theoretical model. We documented the representation approximations by collecting and comparing image features for a range of analog shape parameters.

For the accuracy evaluations, we used the same software libraries as before but sub-selected image features for which we could generate reference feature values. Given the fact that we know the reference value, we could compute normalized relative errors E_i^r per ROI with respect to the reference feature value R_i (in comparison to the minimum value used in Eq. (1)).

$$E_i^r = |V_i - R_i|/R_i \quad (3)$$

V_i is the measured feature value, and i is the index of a ROI (image segment).

A. Test Images

We designed synthetic images with objects for which intensity, shape, and texture could be computed theoretically. Figure 4 shows three such image examples. The mathematical models include linearly increasing intensities with varying ranges and an ellipse shape with varying minor and major axes. The parameters of each model were varied when generating synthetic images. Additional synthetic images were created for testing Euler number as shown in Figure 7.

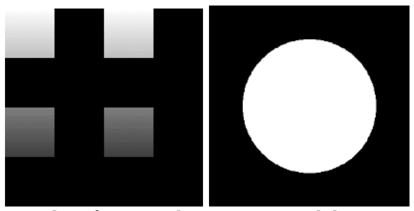


Figure 4: Examples of two synthetic images with known intensity and shape image features.

B. Image Feature Accuracy Analysis

We report accuracy analyses for three image features including (1) perimeter of a circle, (2) major and minor axes of ellipse, and (3) Euler number of a binary image.

Perimeter: We created 23 synthetic binary images of a circle with radius ranging between $r \in [2, 222]$ pixels. The circle generation is done on images with size (500, 500) pixels using the following formula:

$$if\ (x - x_c)^2 + (y - y_c)^2 \leq r^2\ then\ p(x, y) = 1 \quad (4)$$

The circle centroid coordinates are $x_c = 249$ and $y_c = 249$, and r is the circle radius. The perimeter reference value was set to $2\pi r$. Figure 5 displays the normalized perimeter error E_i with respect to the ground truth that is computed as a function of circle radius. Based on Figure 5, the error is large (up to 25%) for small radius values and it converges to a value of 5% as the radius reaches values larger than 100.

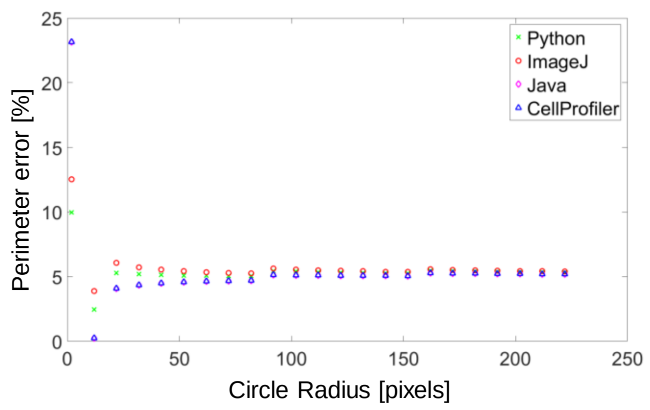


Figure 5. Normalized perimeter error vs circle radius

Major and minor axis length: During the feature variability evaluation, we detected minor differences that were below the 1 % threshold on feature error. To test major and minor axis lengths we created a set of 55 ellipse images with multiple values for major and minor axis length that ranges between 10 and 370. Figure 6 shows the normalized relative error E_i of major axis length computed according to Eq. (3) for ROIs in the 55 simulated images. It was observed that all three implementations had an error larger than 0.1 % when the ellipse shape was flat or the ellipse area was small. All implementations demonstrated the same dependency of feature error on ellipse shape/area.

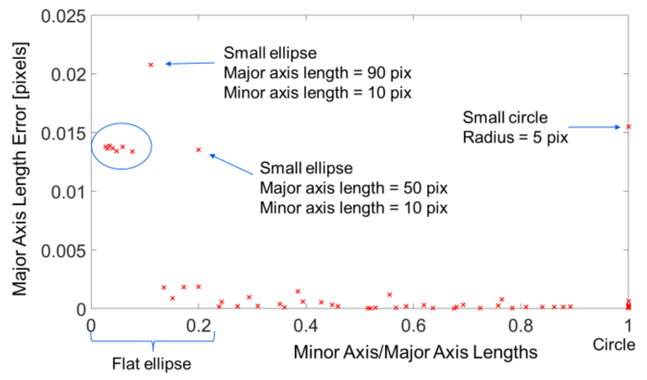


Figure 6. Major axis length normalized relative error as a function of minor/major axis length for 55 synthetic ellipses.

Euler number: Figure 7 displays three synthetic images and their Euler numbers (EN). EN is computed as the number of ROIs minus the number of holes. The values computed by Python deviated from the reference numbers since the implementation assumes only one ROI.

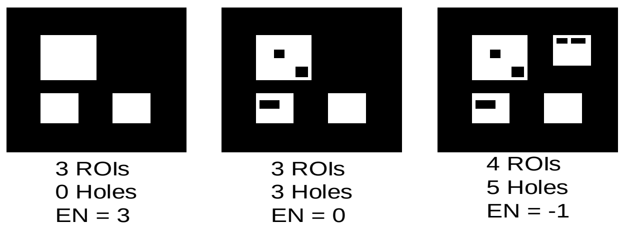


Figure 7. Synthetic images created for testing the Euler Number feature

C. *Discussion*

The design of a synthetic image generator plays a significant role in representing the theoretical value and is always limited by the integer image lattice. For example, the results in Figure 5 and Figure 6 would have been different if we placed the center of each ellipse at the lattice intersection as opposed in the middle of a square pixel (offset is 0.5). While the tools would still agree amongst each other, there will be bias between the computed major axis length value and the reference value. Thus, additional accuracy evaluations are required to determine error contributions of synthetic image generators.

IV. Traceability of Image Features

In the effort to provide access to traceable image features, we designed a client-server architecture of a web system shown in Figure 8. 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 upload collections and download feature values, (3) configuration and execution interface to image features registered in the system, and (4) collaborative access to traceable image feature values that are hyperlinked to their provenance information and all downloadable re-execution artifacts.

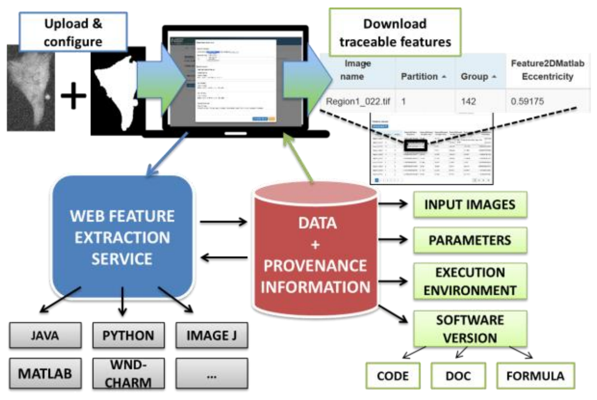


Image name	Partition	Group	Feature2DMatlab Eccentricity
Region1_022.tif	1	142	0.59175

Figure 8: Architecture of a client-server system for traceable image feature extraction

The system was developed as a client-server application, built on top of a scientific workflow management system (WMS). The server-side follows the Java Representational State Transfer (REST) API built using the Spring framework. It is coupled with a MongoDB database and calls Pegasus WMS [8] to manage the feature extraction jobs. The client-side is a light web application written in JavaScript using the AngularJS framework. The client side consumes the REST API from the server side.

A. *Integrating heterogeneous image software*

Ideally, one would like to load an image and its mask to RAM and then compute a spectrum of image features while capturing the provenance information. However, when dealing with image feature methods written in heterogeneous programming languages, the challenges lie in retrieving, compiling and executing feature methods on various platforms, and sharing image data loaded in RAM with each executable. Our approach is to regroup several existing image processing software packages into a single access point by designing a software integration framework. The other challenge of sharing data across heterogeneous software is addressed by using a common file interface to disk instead of more complicated sharing in RAM.

Input and output standardization: Our first step toward the homogenization of heterogeneous software packages was to define standard input and output formats.

Input: We chose the Extensible Markup Language (XML) to design a standard input interface defining the input parameters, such as the input file locations (raw images, optional segmentation masks, and optional tiling masks), output location, and feature extraction configuration (features to extract and their optional parameters values). Our choice of XML was motivated by the fact that most programming languages offer libraries that can be used for manipulating XML documents and the XML format is human-readable.

Output: We chose Comma-Separated Values (CSV) files as the standard output format for the extracted feature values. This format is both machine and human readable and supported in multiple software libraries. In order to merge feature values from diverse feature libraries, we defined naming conventions for the output file names and their CSV headers with input image, ROI and feature information.

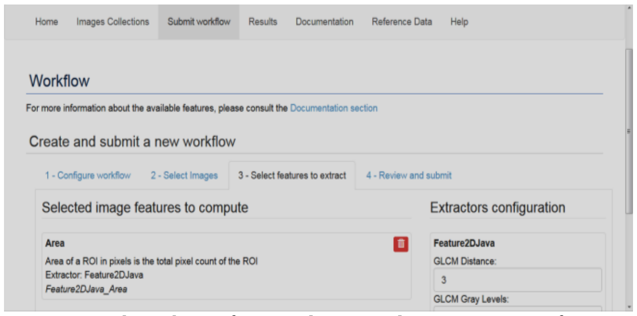
Software executable and metadata integration: The system was designed to be extensible to any image feature implementation that can be run from a command-line. Each software is compiled to an executable that runs on a server and is launched with the common XML input file as an argument. In order to be integrated in the system, software has to be able to read the feature execution inputs from the XML input file and to write the feature results following the conventions we designed for the CSV output format and feature names.

The integration of a new *software executable* in the system is done by adding an entry in the Pegasus Workflow Transformation Catalog. The software is then registered in the MongoDB database by using the system API and saving its metadata in the database. The integration of new *software metadata* is performed by providing a list of features that the software is capable of extracting, along with eventual configurable options. The system can manage several versions of the same software, and each version corresponds to a new metadata entry in the database, linked to the corresponding executable via the Pegasus Transformation Catalog. The metadata stored in the database are used to generate the software documentation and feature selection web interfaces in the system, and to link the feature values to their provenance information.

B. *Key aspects of image feature execution*

In order to facilitate the execution of traceable image features implemented in heterogeneous software, we focused on (1) automated construction of the execution inputs captured by the XML input file, (2) input and output data management, (3) computational scalability of feature extraction, and (4) traceability of computed image feature values.

Construction of feature execution inputs. The input XML file is constructed automatically by collecting inputs via the client interfaces. Figure 9 shows one of the web interfaces to select features. The web interfaces allow a user to upload image and segmentation mask collections, select data and image feature implementations, and then launch the feature extraction. The selected features are linked to their software version metadata and the image collections involved in the extraction jobs are automatically locked on the server to prevent any further modification and allow the provenance information to be persistent.



Workflow

For more information about the available features, please consult the Documentation section

Create and submit a new workflow

Selected image features to compute

Area
Area of a ROI in pixels is the total pixel count of the ROI
Extractor: Feature2DJava
Feature2DJava_Area

Extractors configuration

Feature2DJava
GLCM Distance:
3
GLCM Gray Levels:

Figure 9: The web interface to selecting and extracting image features.

Data management: All input images, computed image features, job configurations and provenance information are stored in a MongoDB database. This database is accessed via the Java API on the server, allowing the registration of new software or new versions of existing software in the system and linking each computed feature value to all the provenance information.

Scalability of computations: The feature extraction computation is managed via the Pegasus scientific workflow [8], which was configured to use the High Throughput Computing workload management system HTCondor. Pegasus distributes the computations depending on the available computational resources, collects computational provenance, and monitors the execution.

Traceability of image feature values: The resulting image feature values from each software package are merged into a single table and delivered in a web interface. The delivered feature values are hyperlinked with the input images, the XML file that was constructed, the executable that was launched, the software repository with the version of the image feature extraction code, the execution environment provenance information, and the web documentation that contains the mathematical formula implemented in the code.

C. *Analysis of image feature traceability*

Image feature values are hyperlinked in the web interface with all artifacts in order to deliver feature traceability as defined in [12] (i.e., "ability to relate artifacts created during the development of a software system to describe the system from different perspectives.").

TABLE II. COMPLEXITY OF FEATURE RE-EXECUTION FROM ARTIFACTS FOR FOUR SOFTWARE PACKAGES. [JRE: JAVA RUNTIME ENVIRONMENT]

Image Feature Software	Complexity of Feature Re-execution	Required Additional Installation
In-house Java	Simple	JRE 1.7 or higher
ImageJ plugin	Medium	JRE 1.7 or higher and ImageJ or Fiji
Python program	Complex	Several libraries and packages with specific dependencies on a chosen operating system
CellProfiler wrapping program	Complex	Several libraries and packages, and CellProfiler v2.1.1 with specific dependencies on a chosen operating system

Beyond describing each feature value with artifacts, our ultimate test of feature traceability is to reproduce any feature value within a tolerance range from the hyperlinked artifacts [13]. This can be easily achieved by replicating the inputs (data and XML input file) and re-launching the feature computation in the same server environment. However, in order to run the downloaded executable with all inputs on a third party computer, one has to replicate the software environment of the server for launching the computation (i.e., install run time engines, scripting environments, libraries, and so on). We tested and classified feature re-execution from traceable artifacts outside of the server according to the complexity of additional installations as: simple, medium and complex. TABLE II. shows such evaluations of the three analyzed software packages.

D. *Analysis of Feature Computation Efficiency*

Having a client-server system for computing traceable image features provided us with a platform for comparing values across software libraries and against reference values as reported in Sections II and III. Here, we added a comparison of feature execution efficiency to complete the feature characterization.

We use 238 microscopy images of NIH 3T3 cells together with their manual segmentation masks from the set described in Section II.B. The raw images and masks are 16-bit per pixel images, each image has a file size of 446 kB and the entire data set contains a total of 8162 ROIs. In order to measure the performance of software packages against large images, we used one large field of view of stem cell colonies (22K x 22K pixels) imaged in phase contrast (16 BPP, 355.9 MB) and in Green Fluorescent Protein (32 BPP after calibration, 1.9 GB). The corresponding segmentation mask (16 BPP, 9.4 MB containing 200 ROIs) was obtained by segmenting the phase contrast image [14].

The following overlapping image features were calculated by all four software packages: Area, Perimeter, Mean intensity and Centroid. The calculations were run on a server with the specifications: Linux Virtual Machine (VM), operating Ubuntu 12.04 LTS 64-bit, four Intel Xeon CPUs and 16GB of RAM. The same calculations were run five times, and the timing and memory results were averaged.

The time and memory consumption breakdown per software is shown in TABLE III. as collected by the Pegasus workflow engine. CPU time is the time measured in system clock ticks for which the CPU was dedicated to feature calculations. This number is converted to seconds for our server with the number of clock ticks per second being 100 Hz. Max RSS (maximum Resident Set Size) is a measure of the memory occupied by a process and held in the main memory (RAM), given in kB (kilobytes) and converted in the table below in MB (megabytes)

Based on TABLE III. the in-house Java package has the fastest CPU computation time for a large number of small images (8162 ROIs in 238 images) with the speed-up factor of 2.81 (Python) and 36.55 (ImageJ). However, against a large image (GFP), Python is the most efficient, with a factor of 3.92 (ImageJ) and 2.51 (Java). We were not able to obtain results for CellProfiler due to its memory consumption exceeding our 10GB limit. Python is also the most efficient in terms of memory consumption, especially for the 3T3 dataset.

TABLE III. AVERAGE CPU TIME, RUN TIME AND MAXIMUM MEMORY CONSUMPTION OF FOUR IMAGE FEATURE CALCULATIONS FOR 8399 ROIS ON 3T3 IMAGES (3T3), 200 ROIS ON PC IMAGE (PC), AND 200 ROIS ON GFP IMAGE (GFP) USING FOUR SOFTWARE PACKAGES.

Software	**Dataset**	**CPU time [s]**	**Run time [s]**	**Max RSS [MB]**
Python	3T3	48.82	58.00	49.98
	PC	35.71	36.38	3,731.41
	GFP	53.78	56.39	4,735.54
ImageJ	3T3	76.09	76.54	312.40
	PC	70.94	48.77	3,042.66
	GFP	211.12	183.66	5,070.66
Java	3T3	17.32	22.88	780.11
	PC	71.73	64.44	6,594.35
	GFP	135.38	186.02	7,292.10
CellProfiler	3T3	633.11	676.99	353.23
	PC	N/A	N/A	N/A
	GFP	N/A	N/A	N/A

V. SUMMARY

We quantified numerical variability of 43 overlapping image features across four software packages, determined image accuracy of 6 features with respect to their reference images and mathematical models, compared execution efficiency of 4 image features, and tested feature traceability in terms of complexity of artifact re-execution on third party hardware. These discoveries were enabled by designing a client-server system for integrating heterogeneous image feature libraries, executing feature calculations, and sharing hyperlinked image feature values with computational provenance artifacts. We deployed the system at NIST for internal use and testing.

In the future, we plan to complete the integration of additional publicly available feature extraction software in [1], [2] and [7] in order to make the image features traceable via our designed framework. We will also investigate the variability and accuracy of texture features, and disseminate the software and image feature studies to the image processing communities.

VI. ACKNOWLEDGMENT

We would like to acknowledge Antoine Vandecreme from the Computational Science in Metrology project at NIST who has contributed to the code development of the web image feature extraction system.

VII. 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 the National Institute of Standards and Technology, nor is it intended to imply that the products identified are necessarily the best available for the purpose.

REFERENCES

[1] M. V Boland and 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*, vol. 17, no. 12, pp. 1213–1223, 2001.

[2] N. Orlov, L. Shamir, T. Macura, J. Johnston, D. M. Eckley, and I. G. Goldberg, "WND-CHARM: Multi-purpose image classification using compound image transforms," *Pattern Recognit. Lett.*, vol. 29, no. 11, pp. 1684–1693, Aug. 2008.

[3] A. E. Carpenter, "Extracting rich information from images," *Methods Mol. Biol.*, vol. 486, pp. 193–211, 2009.

[4] "MATLAB and Image Processing Toolbox Release 2015b," *MathWorks Inc.* [Online]. Available: http://www.mathworks.com/help/images/index.html. [Accessed: 16-Mar-2016].

[5] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, "scikit-image: image processing in Python.," *PeerJ*, vol. 2, p. e453, Jan. 2014.

[6] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J.-Y. Tinevez, D. J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, and A. Cardona, "Fiji: an open-source platform for biological-image analysis," *Nat. Methods*, vol. 9, no. 7, pp. 676–682, 2012.

[7] Itseez, "OpenCV (Open Source Computer Vision Library)," 2016. [Online]. Available: http://opencv.org/. [Accessed: 20-Mar-2016].

[8] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Futur. Gener. Comput. Syst.*, vol. 46, pp. 17–35, May 2015.

[9] I. NIST, "Image Features," *web page*, 2016. [Online]. Available: https://isg.nist.gov/deepzoomweb/stemcellfeatures. [Accessed: 31-Mar-2016].

[10] M. Halter, D. R. Sisan, J. Chalfoun, B. L. Stottrup, A. Cardone, A. A. Dima, A. Tona, A. L. Plant, and J. T. Elliott, "Cell cycle dependent TN-C promoter activity determined by live cell imaging," *Cytom. Part A*, vol. 79, no. 3, pp. 192–202, Mar. 2011.

[11] J. Chalfoun, M. Majurski, A. Peskin, C. Breen, and P. Bajcsy, "Empirical Gradient Threshold Technique for Automated Segmentation across Image Modalities and Cell Lines.," *J. Microsc.*, no. Under Review, pp. 1–18, 2014.

[12] G. Spanoudakis and A. Zisman, "Software traceability: a roadmap," in *Advances in software knowledge engineering*, vol. III, S. K. Chang, Ed. World Scientific Publishing, 2005, pp. 1–35.

[13] T. Crick, B. A. Hall, and S. Ishtiaq, "Reproducibility as a Technical Specification," *Comput. Res. Repos.*, vol. abs/1504.0, p. 6, 2015.

[14] K. Bhadriraju, M. Halter, J. Amelot, P. Bajcsy, J. Chalfoun, A. Vandecreme, B. S. Mallon, K. Park, S. Sista, J. T. Elliott, and A. L. Plant, "Large-scale time-lapse microscopy of Oct4 expression in human embryonic stem cell colonies," *Stem Cell Res.*, vol. 17, no. 1, pp. 122–129, 2016.