SoftwareX 16 (2021) 100863

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Imaging framework: An interoperable and extendable connector for image-related Java frameworks

Christoph Praschl [a,*], Andreas Pointner [a], David Baumgartner [c], Gerald Adam Zwettler [a,b]

[a] Research Group Advanced Information Systems and Technology, Research and Development Department, University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg i. M., Austria

[b] Department of Software Engineering, School of Informatics, Communications and Media, University of Applied Sciences Upper Austria, Softwarepark 11, 4232 Hagenberg i. M., Austria

[c] Department of Computer Science, Norwegian University of Science and Technology, Høgskoleringen 1, Trondheim, Norway

ARTICLE INFO

Article history:
Received 7 July 2021
Received in revised form 22 September 2021
Accepted 19 October 2021

Keywords:
Java
Image processing
Computer vision
Interoperability and extendability

ABSTRACT

The number of computer vision and image processing tasks has increased during the last years. Although Python is most of the time the first choice in this area, there are situations, where the utilization of another programming language such as Java should be preferred. For this reason, multiple Java based frameworks as e.g. OpenIMAJ, ND4J or multiple OpenCV wrappers are available. Unfortunately, these frameworks are not interoperable at all. In this work, the open-source Imaging Framework is introduced to solve exactly this problem. The project features a concept for combining multiple frameworks and provides an interoperable and extendable foundation to 9 image-related projects with 10 different image representations in Java.

Code metadata

Current code version	1.2.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00126
Legal Code License	Mozilla Public License, v. 2.0
Code versioning system used	GIT
Software code languages, tools, and services used	Java
Compilation requirements, operating environments & dependencies	JDK 11 or newer
If available Link to developer documentation/manual	https://fhooeaist.github.io/imaging/
Support email for questions	contact@aist.science

1. Motivation and significance

This work introduces a framework that encourages the interoperability of Java frameworks in the context of image-related tasks. With the steady progress in the field of machine learning (ML), the number of computer vision (CV) applications based on methods as convolutional neural networks are boosted, too. Besides, the importance of image processing (IP) tasks such as pre-processing or data augmentation has grown significantly in the last years. Hand in hand with this, also the quantity of frameworks that are used in this context has rose and existing frameworks have been extended continuously. Especially, frameworks such as Tensorflow [1], OpenCV [2], PyTorch [3] and similar ones are based on the programming languages Python and C++ and have gained strongly in popularity. In particular, Python has grown more and more in this area in the recent years [4]. But this domain is not limited to these programming languages and the number of frameworks in other languages as e.g. Java has also expanded steadily.

For this reason, image processing tasks among Python and Java were compared using plain implementations without additional frameworks as well as framework based comparisons [5]. These

* Corresponding author.
E-mail addresses: christoph.praschl@fh-hagenberg.at (Christoph Praschl), andreas.pointner@fh-hagenberg.at (Andreas Pointner), david.baumgartner@ntnu.no (David Baumgartner), gerald.zwettler@fh-hagenberg.at (G.A. Zwettler).

https://doi.org/10.1016/j.softx.2021.100863

comparisons show that Python itself is not really suitable for this area of application without using additional frameworks, due to the high performance impact of basic control structures like loops as investigated by Rui et al. [6]. This performance impact is also the motivation of Kwan Lam et al. [7] for the creation of the Numba Just-In-Time compiler for Python. Most common Python frameworks in the discussed area of application are based on a foundation written in C++, which also allows implementing GPU-accelerated methods due to the usage of native libraries as CUDA [8] or OpenCL [9]. But C++ comes with a trade-off as programs and libraries implemented in this programming language have to be built in reference to the target platform and are for this not always available or suitable for certain applications. This is also a transitive problem for Python, if such frameworks are used. In difference to that, Java focuses on its platform independence. The before mentioned comparison of Java and Python also showed, that Java is highly suitable with only a small loss of performance in domains, where e.g. GPU support is not required, certain C++ libraries are not available or where the target application is already based on Java. It also showed that the usage of a native C++ library as OpenCV comes with a lower overhead in Java compared to Python, at least in the presented use case of applying convolutional filters.

Based on this situation you can use different pure Java frameworks for ML-, CV- or IP-related tasks as ImageJ [10], Nd4j [11] or OpenIMAJ [12], or native wrappers of popular C++ libraries as OpenCV, Tensorflow or Tesseract [13]. While many Python frameworks/wrappers as OpenCV, Tensorflow or PyTorch are based on or at least support the library numpy [14], the Java equivalents use all their own foundation and domain classes. Like this, the mentioned Java frameworks are not compatible and cannot be combined in a comprehensive way. This is the motivation of the Imaging Framework: an interoperable and extendable connector for image-related Java frameworks.

To achieve this interoperability, the Imaging Framework is based on a module structure with the `API` module as central component, cf. Fig. 1. Based on this main module, additional internal features as basic image processing or mesh related functions are provided via the `Core` and the `Mesh` modules. In addition to these two internal modules, there are multiple connection modules used to integrate external frameworks. These extensions provide framework specific implementations of the most important interfaces defined in the `API` module and that way establish interchangeable objects, respectively images and image related functions. The project is already used in various domains of vision-based analytics. One example is the utilization of the framework in the context of image-based determination of the orientation of augmented reality devices in outdoor scenarios [15]. Next to that, the project is used for the pose estimation of human bodies [16], as well as the transformation of human mask [17]. In addition to these use cases, the Imaging Framework is also used for the 3D reconstruction of building plans [18], and for automated person identification in online banking [19].

2. Software description

The Imaging Framework is a free Java framework published under the Mozilla Public License Version 2.0 [20] and is open for contributions on GitHub [21]. It is available via Apache Maven [22] on Maven Central [23] and can like this be easily integrated into any Maven based Java application. A general framework documentation can be found in [24], while the Java class documentation can be found in [25]. The original purpose of the Imaging Framework was to combine image-processing tasks using OpenCV in Java with functionalities of Microsoft's Cognitive Services [26] for face recognition and Tesseract [13] for optical character recognition. Since then, the number of supported

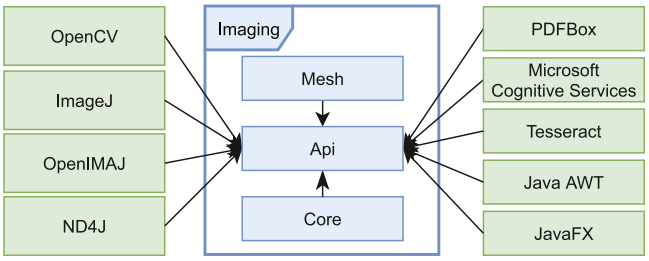

Fig. 1. Module structure of the imaging project with the API module as central component and additional modules building up on that.

frameworks has grown to 9 frameworks and 10 image representations, that can be used interchangeable, cf. Fig. 1. At the time of writing, the Imaging Framework supports interoperability between the following frameworks:

1. OpenCV
2. ImageJ
3. OpenIMAJ
4. ND4J
5. PDFBox [27]
6. Microsoft Cognitive Services
7. Tesseract
8. Java AWT [28]
9. JavaFX [29]

2.1. Software architecture

The Imaging Framework is built around one generic core interface of the `API` module called `ImageWrapper`, see Fig. 2. This interface wraps the most important information related to an 2D image: (I) the width, (II) the height, (III) the color model with the associated number of channels and (IV) the raw image itself. To avoid memory leaks, especially with natively allocated objects, the `ImageWrapper` interface extends Java's `Autocloseable` and like that is usable in try-with-resources blocks.

In the context of region of interests, the Imaging Framework provides another domain class, that is called `SubImageWrapper` and represents a spatial restricted view of a referenced source `ImageWrapper`. Any read or write access to such a region of interest is propagated to the originating `ImageWrapper`. The idea of the `ImageWrapper` interface is to provide framework specific implementations, which can be used interoperable due to the interface. To reinforce this programming paradigm, implementations of the interface are package protected and are only available via an associated `ImageFactory`. For this reason, the `ImageFactory` interface provides multiple functionalities to create a new image based on different parameters. The actual `ImageFactory` implementations are in turn package protected. To be still able to use an `ImageFactory` for a given generic image type, its implementations are registered to Java's `ServiceLoader` [30] and can be accessed via the central `ImageFactoryFactory` class. This encapsulation allows to keep an user of the framework away from implementation details, since only implementations against the interfaces are required and accessible.

Due to the strict separation of the architecture, an image-related framework can be connected to the Imaging Framework and, like that, achieving interoperability to the other supported frameworks with only three steps. To do so, an implementation of the `ImageWrapper` interface, respectively an extension of the `AbstractImageWrapper` implementation, must be created in a first step. For this, you have to implement the following five methods according to your image representation: (I) `getHeight()`, (II) `getWidth()`, (III) `getValue()`, (IV) `setValue()` and (V) `getSupportedType()`. Next to the `ImageWrapper`, you have to provide an `ImageFactory` implementation that allows

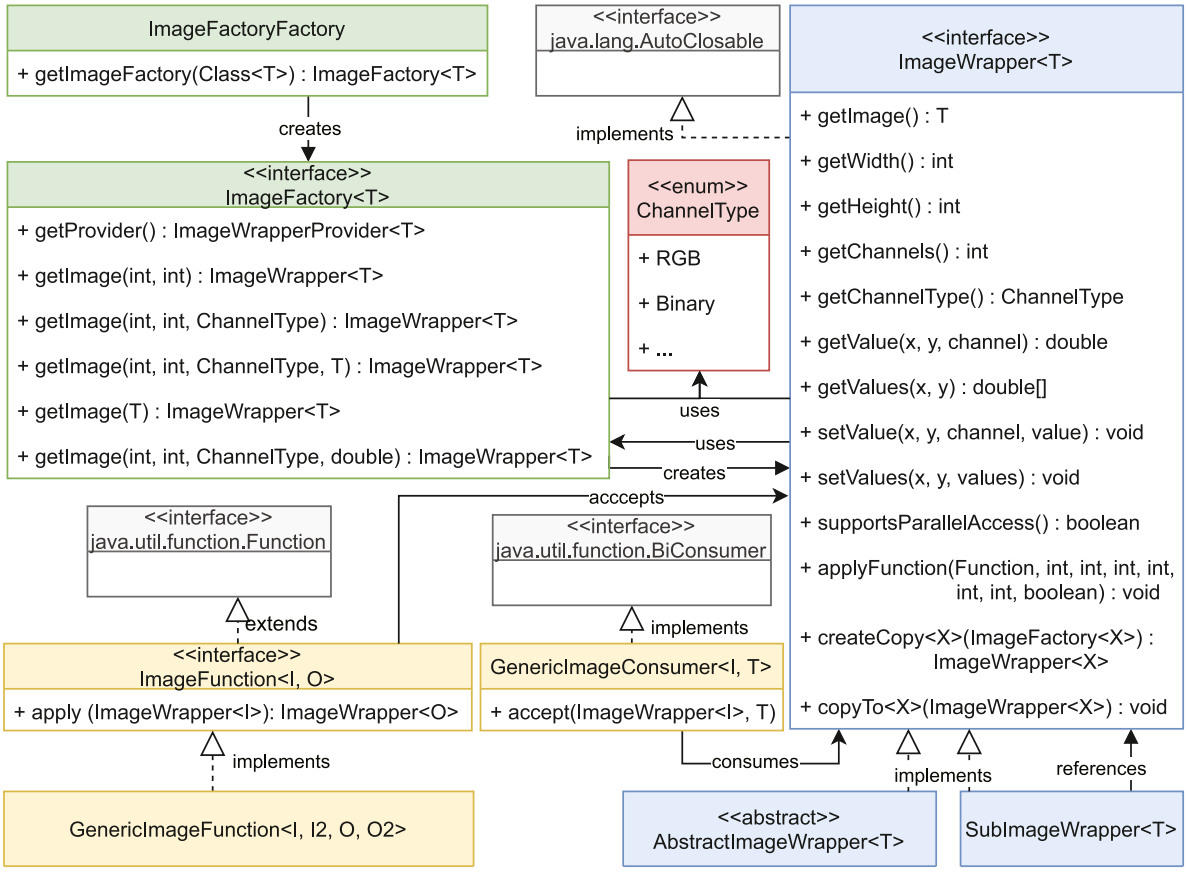

Fig. 2. Base classes and interfaces of the imaging project build around the `ImageWrapper` interface. Blue entities represent the image-related core classes of the Imaging project, while green ones are factories to create those. In addition to that yellow entities are functions and consumers to interact with images. The gray entities represent important Java base interfaces. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

to create images of your representation in a second step. Finally, you have to register your factory to Java's `ServiceLoader`.

The Imaging Framework also provides image-related implementations for Java's `Function` and `BiConsumer` interfaces. On the one hand the `ImageFunction` interface is the base for image manipulating tasks in a functional-like programming style that uses a given image to create a new one, on the other hand the `GenericImageConsumer` can be used to manipulate a given image based on an additional object. While the first variant can be utilized to implement filters like a Gaussian blur where the source as well as the result image are required in subsequent steps, the last one can for example be used to draw geometric figures as a circle onto a given image.

Next to the image-related classes, the Imaging Framework also provides a basic geometric domain architecture, that is used for different aspects of the framework, cf. Fig. 3. These geometric classes are built around the central `AbstractJavaPoint` class. This base class is extended by `JavaPoint2D` as well as, `JavaPoint3D` and is used to represent any two- or three-dimensional point. Based on this spatial information, different abstract collection classes are used to represent e.g. an unsorted point cloud using `AbstractJavaPointCloud`, or a sorted point sequence in the form of a polyline or polygon, that is called `AbstractJavaPolygon`. Next to these first two abstract classes, there is also the abstract representation of a line between two points called `AbstractJavaLine`. The four abstract classes in the geometric domain are extended by different two- or three-dimensional implementations. These classes follow a programming paradigm of rich and unmodifiable domain classes. This means that the implementations offer, multiple functionalities and cannot be manipulated after construction, but require the creation of new objects on value changes. As example for the rich class paradigm, the two-dimensional line implementation allows to calculate its length or the intersection with another line and will always consist of the same two points due to its unmodifiable characteristic. In the case that one of the associated points should be changed, a new line object has to be created containing the new point and the remaining old point.

2.2. Software functionalities

Next to the meta information, the `ImageWrapper` interface also provides common functionality to image related tasks as getting and setting a pixel's value at a given position, or applying a pixel based read or write function. These functions are the fundament of the interchangeability of images between different frameworks via the Imaging Framework and are used to create a deep copy-mechanism as well as pixel manipulation functionalities. This copy functionality is implemented with two methods (I) `createCopy()` and (II) `copyTo()` of the `ImageWrapper` interface. This is also the basis of the `GenericImageFunction` that extends the interoperability of the Imaging Framework and allows to apply a function for an image of type *A*, using a temporary image of type *B* if required. The wrapped function results in an image of type *C*, which is cast to type *D* if necessary, see Fig. 4. While the `GenericImageFunction` comes with the big advantage according to the usability and the interoperability of different frameworks, it comes with the trade-of, that images are

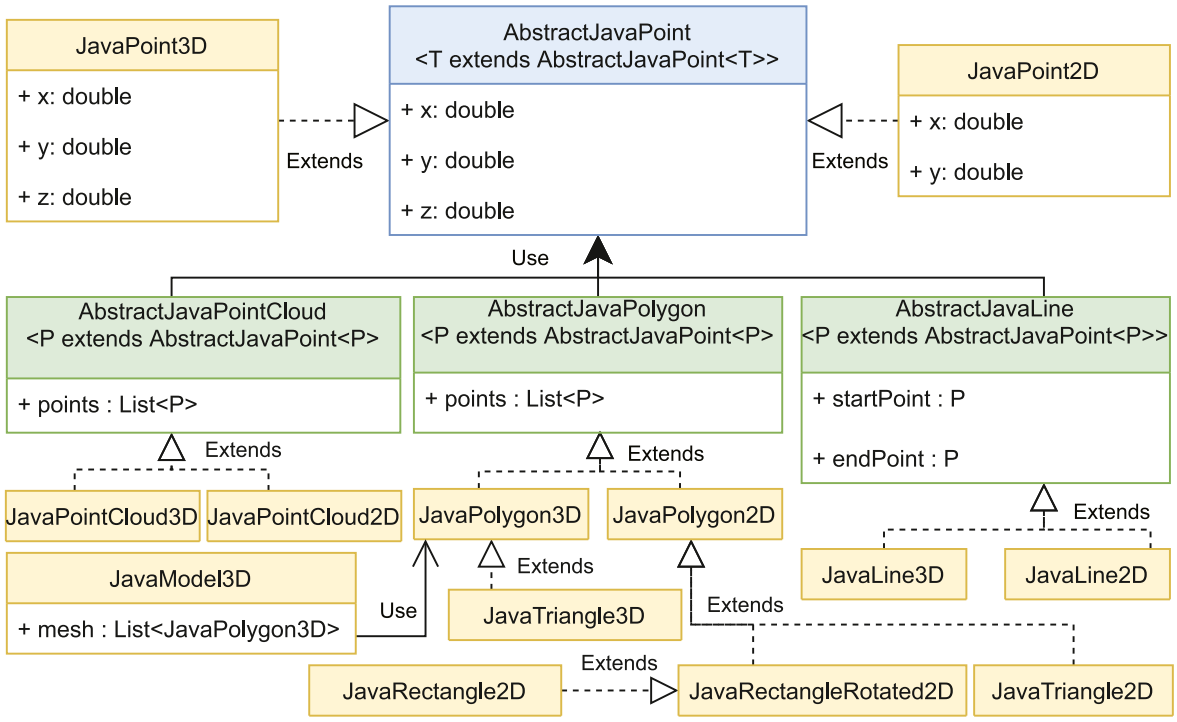

Fig. 3. Geometric domain architecture of the imaging project used to represent different geometric forms. The architecture is build around the central `AbstractJavaPoint` class (blue) and other abstract classes building on that (green). There are multiple two- and three-dimensional implementations of the abstract base classes, which are used at different positions of the Imaging Framework. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

pixel-wise cast. So in the worst case with neither a matching input, nor a matching output type, there is a runtime overhead of $O(2 \times w \times h)$ in reference to the image's width w and its height h. In extreme cases, where many functions of another framework are used, the recommended process consists of (I) casting the image once, applying the functions and finally casting the result back to the expected target or (II) wrapping multiple, consecutive function calls inside the `GenericImageFunction`, cf. Listing 2.

Based on the described fundamentals, the core module provides image processing functionalities as different high and low pass filters, as well as arithmetic operators for e.g. adding, subtracting or multiplying images. Amongst those operators also some basic functionalities are contained as interpolation methods, fitness functions, segmentation methods or e.g. drawing consumers. These implementations are part of the `Core` module. The reason for this module, is to provide some basic functionality in cases, where mostly GUI related tasks and only some basic image processing functionalities are required in an application and other dependencies would lead to an unnecessary overhead.

3. Illustrative examples

In a first example [31], two typical base steps using the Imaging Framework are shown, cf. Listing 1. For this, in a first step the `ImageFactoryFactory` is used to get an actual `ImageFactory` for a given type, namely `short[][][]`. This array is the simplest representation in the Imaging Framework for an image and uses the first array dimension for the image's columns, the second dimension for the rows and the last dimension for the actual pixel information in the form of a channel array. Using the `getImage()` function allows to create such an image with a size of 100 × 100 pixels and a grayscale channel type as color model. The so created image is used as input for an image consumer in a second step using `DrawCircle`, which allows to decorate a given image with a circle based on a position represented as `JavaPoint2D` using the `accept()` method. The manner how the circle is drawn in terms of e.g. the color is controlled via different set-methods.

Listing 1: Simple example for creating a new grayscale image with 100x100px, onto which a white circle is drawn at the position (50,50) using a DrawCircle consumer.

```
1  /* (1) Create a new short-array based
       image */
2  try(var image = ImageFactoryFactory
3          .getImageFactory(short[][][].
               class)
4          .getImage(100, 100, ChannelType
               .GREYSCALE)){
5
6      /* (2) Draw a circle on the image
           */
7      var draw = new DrawCircle<short
           [][][]>();
8      draw.setRadius(3);
9      draw.setColor(new double[]{255});
10     draw.accept(image, new JavaPoint2D
           (50, 50));
11 }
```

The second example illustrates the interoperability of the Imaging project based on the `GenericImageFunction`, cf. Listing 2. For this, a random image is created in a first step. This process is comparable to the first step of the previous example, but uses the `getRandomImage()` function instead of `getImage()`. This method is provided primarily for testing and allows to create a randomly initialized image within a given range, in the presented case with 8-bit unsigned grayscale values in the range of [0; 255]. Note the `@Cleanup` annotation of Project

Lombok [32], that is used to avoid boilerplate code due to try-with-resources-blocks, as shown in the first example. It allows to ensure the memory deallocation of the wrapped object at the end of the scope. In a second step, the actual function is created using a Canny edge detector and a subsequent dilation, both implemented with the AistCV [33] Java build of the OpenCV framework. This function is wrapped in the third step using the `GenericImageFunction`. When creating the `GenericImageFunction` object, the interim image type of the wrapped function using OpenCV's `Mat` class and the expected result type using a three-dimensional double array are defined. In the fourth and final step of the example, the wrapped function is applied, that is implemented using OpenCV, on an image represented with OpenIMAJ's `FImage` class. The so created dilated edge representation of the input image is cast to the expected `double[][][]` representation by the `GenericImageFunction` object.

Listing 2: Example for the interoperability of the Imaging Framework using a Canny edge detection together with a image dilation implemented in OpenCV, that is applied on an image created with OpenIMAJ and resulting in a image represented as double array.

```
/* (1) Create new OpenIMAJ image */
Random rand = new Random(768457);
@lombok.Cleanup
var input = ImageFactoryFactory
        .getImageFactory(FImage.class)
        .getRandomImage(100, 100,
            ChannelType.GREYSCALE, rand
            , 0, 255, false);
/* (2) Combine multiple OpenCV function
     calls in one lambda function using
     a Canny edge detector and an image
     dilation */
ImageFunction<Mat, Mat> f = i -> {
    var image = i.getImage();
    var res = new Mat();
    // apply canny edge detector
    Imgproc.Canny(image, res, 15, 125);
    // prepare dilation
    var element = Imgproc.
        getStructuringElement(
            Imgproc.CV_SHAPE_RECT,
            new Size(3,3),
            new Point(1, 1)
    );
    // apply dilation
    Imgproc.dilate(res, res, element);
    return ImageFactoryFactory
            .getImageFactory(Mat.class)
            .getImage(i.getHeight(), i.
                getWidth(),
                    ChannelType.BINARY,
                        res);
};
/* (3) Wrap the function in a
    GenericImageFunction */
var function = new GenericImageFunction
    <FImage, double[][][], Mat, Mat>(f,
     Mat.class, double[][][].class);
/* (4) Apply the function on a non-
    OpenCV image and create an image
    represented as double-array */
@lombok.Cleanup
var thresholdResult = function.apply(
    input);
```

4. Impact

Currently, the Imaging Framework supports images represented as three-dimensional `short` or `double` arrays, as well as Java AWT's `BufferedImage`, JavaFX's `WriteableImage`, OpenCV's `Mat`, ND4J's `INDArray`, OpenIMAJ's `FImage` and `MBFIMage`, as well as ImageJ's `ImageProcessor` and `ImageStack`. For this, it allows to use the mentioned frameworks completely interoperable and can be easily extended with further ones. Based on this characteristic, the Imaging Framework represents a step towards closing the gap between image-related Java frameworks, which originates in the situation, that most of the mentioned frameworks provide a partially overlapping functionality, but focus on certain areas of application without any common basis. The Imaging Framework allows picking out the required features from certain frameworks and to combine those to create more advanced functionalities in an easy and comprehensive way.

Actually, the Imaging Framework allows creating a GUI application using e.g. AWT, Swing, JavaFX or even ImageJ and to add image-related methods using additional image processing frameworks such as OpenCV. This is achieved due to the support of the image representations of the mentioned frameworks within the imaging environment. Like this, any of the named GUI frameworks can be used for frontend related tasks and any supported image processing framework can be used for image-related backend tasks.

Next to that, the Imaging Framework supports the combination of computer vision tasks with classic image processing approaches. While, for example ND4J focuses on the utilization of deep learning models such as neural networks, it does not provide advanced image processing methods. To overcome this limitation, frameworks as OpenCV or OpenIMAJ can be used. The gap between these frameworks is in turn closed by the presented Imaging Framework.

Due to the support of ND4J, the Imaging Framework does not only close the gap between Java and computer vision tasks, but also between GPU acceleration and image-related Java based applications. Since ND4J comes with CUDA support and is one of the connected frameworks, it also increases the interoperability according to the supported platforms and can have a huge impact according to the performance of many image-related tasks. Next to ND4J, there are also Java based wrappers for OpenCV like JavaCV [34], that come with GPU support and can be easily added to the imaging environment.

Furthermore, due to the provided interfaces and service infrastructure the Imaging Framework allows to be easily extended by additional frameworks, requiring only a few steps, as described in Section 2.1. Like this, it is possible to increase the interoperability over further frameworks, by integrating the associated image representations using an individual implementation of the `ImageWrapper` and `ImageFactory` interfaces.

5. Conclusions and outlook

In this work, the Imaging Framework is presented, an interoperable and extendable connector for image-related Java frameworks. It was developed to close the gap between multiple Java frameworks related to image processing and computer vision tasks. The focus of the project is to provide a concept on combining multiple frameworks without requiring any knowledge in terms of differences of the used image representations, and to provide an interoperable mechanism across framework boundaries.

Due to the increasing number of image-related tasks in the context of image processing and especially computer vision in the previous years, our framework reaches a broad community. Although Python is most of the time the first choice in this area of application, there are situations, where developers and researchers can profit from the advantages of executing image processing tasks in Java. To our knowledge, there is no comparable project available in the Java ecosystem.

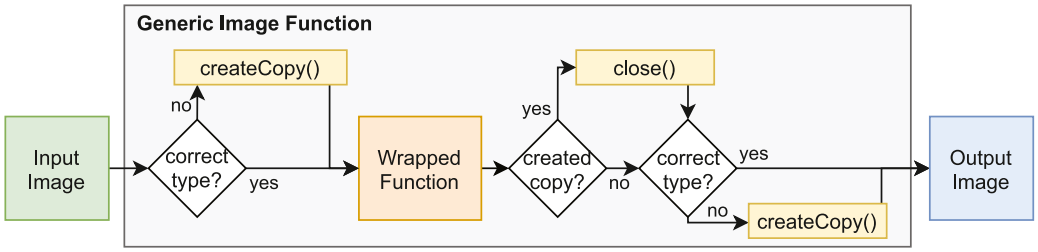

Fig. 4. Sequence of a generic image function with a copy-based casting of the image if required before and after applying the wrapped function.

The future development of the imaging project is dedicated to increasing the number of supported frameworks as Tensorflow and JavaCV amongst others. In addition to that, it is also planned to increase the number of core functionalities and to further improve the code quality according to features, bugs and test automation.

CRediT authorship contribution statement

Christoph Praschl: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing. **Andreas Pointner:** Conceptualization, Methodology, Software, Writing – review & editing. **David Baumgartner:** Conceptualization, Methodology, Software, Writing – review & editing. **Gerald Adam Zwettler:** Conceptualization, Supervision, Writing – review & editing, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. It was carried out to exploit synergies from the research projects Guide (FFG project number 859431), Drive4Knowledge (FFG project number 862975), PASS (FFG project number 872928) and TrueSize (FFG project number 872105) funded by the Austrian Research Promotion Agency FFG.

References

[1] Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, et al. Tensorflow: A system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation. 2016. p. 265–83.
[2] Bradski G, Kaehler A. Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc."; 2008.
[3] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. Pytorch: An imperative style, high-performance deep learning library. 2019, arXiv preprint arXiv:1912.01703.
[4] Raschka S, Patterson J, Nolet C. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. Information 2020;11:193. http://dx.doi.org/10.3390/info11040193.
[5] Praschl C, Pointner A, Baumgartner D. FHOOEAIST/coffee-burns-snake: 1.0.0. 2021, http://dx.doi.org/10.5281/zenodo.4980088.
[6] Pereira R, Couto M, Ribeiro F, Rua R, Cunha J, Fernandes JaP, et al. Energy efficiency across programming languages: How do energy, time, and memory relate? In: Proceedings of the 10th ACM SIGPLAN international conference on software language engineering. New York, NY, USA: Association for Computing Machinery; 2017, p. 256–67. http://dx.doi.org/10.1145/3136014.3136031.
[7] Lam SK, Pitrou A, Seibert S. Numba: A LLVM-based python JIT compiler. In: Proceedings of the second workshop on the LLVM compiler infrastructure in HPC. New York, NY, USA: Association for Computing Machinery; 2015, p. 1–6. http://dx.doi.org/10.1145/2833157.2833162.
[8] Sanders J, Kandrot E. CUDA by example: An introduction to general-purpose GPU programming. Addison-Wesley Professional; 2010.
[9] Stone JE, Gohara D, Shi G. OpenCL: A parallel programming standard for heterogeneous computing systems. Comput Sci Eng 2010;12(3):66.
[10] Rueden CT, Schindelin J, Hiner MC, DeZonia BE, Walter AE, Arena ET, et al. ImageJ2: ImageJ for the next generation of scientific image data. BMC Bioinform 2017;18(1). http://dx.doi.org/10.1186/s12859-017-1934-z.
[11] Team EDD. ND4J: Fast, scientific and numerical computing for the JVM. 2016, URL https://github.com/eclipse/deeplearning4j.
[12] Hare JS, Samangooei S, Dupplaw DP. OpenIMAJ and ImageTerrier: Java libraries and tools for scalable multimedia analysis and indexing of images. In: Proceedings of the 19th ACM international conference on multimedia. New York, NY, USA: ACM; 2011, p. 691–4. http://dx.doi.org/10.1145/2072298.2072421.
[13] Smith R. An overview of the tesseract OCR engine. In: Ninth international conference on document analysis and recognition. vol. 2. IEEE; 2007, p. 629–33.
[14] Oliphant TE. A guide to NumPy. vol. 1. Trelgol Publishing USA; 2006.
[15] Praschl C, Krauss O, Zwettler GA. Enabling outdoor MR capabilities for head mounted displays: a case study. Int J Simul Process Model 2020;15(6):512–23. http://dx.doi.org/10.1504/IJSPM.2020.112463.
[16] Baumgartner D, Zucali T, Zwettler GA. Hybrid approach for orientation-estimation of rotating humans in video frames acquired by stationary monocular camera. In: Computer science research notes. Západočeská univerzita; 2020, p. 39–47. http://dx.doi.org/10.24132/csrn.2020.3001.5.
[17] Zwettler GA, Praschl C, Baumgartner D, Zucali T, Turk D, Hanreich M, et al. Three-step alignment approach for fitting a normalized mask of a person rotating in A-pose or T-pose essential for 3D reconstruction based on 2D images and CGI derived reference target pose. In: VISIGRAPP (5: VISAPP). 2021, p. 281–92. http://dx.doi.org/10.5220/0010194102810292.
[18] Pointner A, Praschl C, Krauss O, Schuler A, Helm E, Zwettler G. Line clustering and contour extraction in the context of 2D building plans. In: Proceedings of the 17th international joint conference on computer vision, imaging and computer graphics theory and applications. Západočeská univerzita; 2021, p. 11–20. http://dx.doi.org/10.24132/csrn.2021.3101.2.
[19] Pointner A, Krauss O, Freilinger G, Strieder D, Zwettler G. Model-Based Image Processing Approaches for Automated Person Identification and Authentication in Online Banking. In: Conference: 30th European modeling and simulation symposium. 2018. p. 36–46.
[20] Mozilla. Mozilla public license, version 2.0. 2013, URL https://www.mozilla.org/en-US/MPL/2.0/, [Accessed 17 June 2021].
[21] Praschl C, Pointner A, Baumgartner D. FHOOEAIST/imaging: 1.2.0. 2021, http://dx.doi.org/10.5281/ZENODO.4541503, URL https://zenodo.org/record/4541503.
[22] Miller FP, Vandome AF, McBrewster J. Apache Maven. Alpha Press; 2010.
[23] Sonatype, Inc. Maven central repository search. 2021, URL https://search.maven.org/search?q=g:%22science.aist.imaging%22, [Accessed 17 June 2021].
[24] Praschl C, Pointner A, Baumgartner D. Imaging – imaging. 2021, URL https://fhooeaist.github.io/imaging/index.html, [Accessed 17 June 2021].
[25] Praschl C, Pointner A, Baumgartner D. Overview (imaging 1.2.0 API). 2021, URL https://javadoc.io/doc/science.aist.imaging/imaging/latest/index.html, [Accessed 17 June 2021].
[26] Del Sole A. Introducing microsoft cognitive services. In: Microsoft computer vision APIs distilled. Springer; 2018, p. 1–4.
[27] Butler S, Gamalielsson J, Lundell B, Brax C, Mattsson A, Gustavsson T, et al. Maintaining interoperability in open source software: A case study of the apache PDFBox project. J Syst Softw 2020;159:110452.
[28] Zukowski J. Java AWT reference. O'Reilly Media; 1997.
[29] Chin S, Vos J, Weaver J. Javafx fundamentals. In: The definitive guide to modern java clients with JavaFX. Springer; 2019, p. 33–80.

[30] Oracle. ServiceLoader (java SE 9 & JDK 9). 2017, https://docs.oracle.com/javase/9/docs/api/java/util/ServiceLoader.html,[Accessed 21 June 2021].
[31] Praschl C, Pointner A, Baumgartner D. FHOOEAIST/imaging-paper: 1.0.0. 2021, http://dx.doi.org/10.5281/zenodo.5005652.
[32] Zwitserloot R, Spilker R. Project Lombok. 2016.
[33] Baumgartner D, Praschl C. FHOOEAIST/aistcv: 4.3.0. 2021, http://dx.doi.org/10.5281/ZENODO.4506232, URL https://zenodo.org/record/4506232.
[34] Audet S. JavaCV: Java interface to OpenCV, FFmpeg, and more. 2018, URL: https://github.com/bytedeco/javacv, [Accessed on 5 March 2020] Archived at http://www.webcitation.org/6hZyxW85u.