ENABLING REPRODUCIBLE OBIA WITH OPEN-SOURCE SOFTWARE IN DOCKER CONTAINERS

C. Knoth[a,*], D. Nüst[a]

[a] Institute for Geoinformatics, University of Münster, Münster, Germany (christianknoth, daniel.nuest)@uni-muenster.de

KEY WORDS: reproducibility, open-source OBIA, conflict damage assessment, containerisation, Docker

ABSTRACT:

While most approaches in Object-Based Image Analysis (OBIA) currently rely on proprietary software, the interest in free and open-source software (FOSS) for OBIA is growing. This interest stems not only from cost savings, but also from benefits concerning reproducibility and collaboration. However, technical challenges hamper practical reproducibility, especially when multiple software packages are involved. In this study, we use the Docker technology to containerise an OBIA workflow in a well-defined FOSS environment. Running the analysis inside a container eliminates the need to recreate the original software environment on the executing computer. We explore the approach using two software stacks (InterIMAGE, and QGIS in combination with Orfeo ToolBox, SAGA and Python libraries) to perform an exemplary analysis detecting destruction of buildings in bi-temporal images of a conflict area. The analysis combines feature extraction techniques with segmentation and object-based analysis to detect changes and to distinguish disappeared buildings from similarly changed non-target structures. The resulting workflow is published as FOSS comprising both the model and a ready to use Docker image including all required software and data. The presented solution advances OBIA in the following aspects: higher transparency of methodology; easier reuse and adaption of workflows; better transferability between operating systems; complete description of software environment; and easy adoption of OBIA workflows by image analysis experts and non-experts.

1. INTRODUCTION

Openness in conducting research is not a new topic, but there clearly is a recent trend enforcing transparency and availability under the terms Open Science[1] and Open Access[2]. All stakeholders in the research process contribute rules, incentives or guidelines to foster openness. For example (a) on the funding side, the EU requires open access as part of Horizon 2020[3] and builds the European Open Science Cloud[4], (b) on the publishing side, journals such as Science (Nosek et al., 2015) and Bioinformatics (Peng, 2009) encourage reproducibility, and (c) researchers themselves argue for reproducibility in "Five selfish reasons to work reproducibly" (Markowetz, 2015) or publish "Ten Simple Rules for Reproducible Computational Research" (Sandve et al., 2013), which essentially argue in favour of a proper scientific workflow simply to be able to reproduce *your own* results. A core notion of all of these activities is the ideal to publish data, methods, and software along with scholarly publications.

A definition of the term reproducibility is far from trivial. It is even used together with other terms to describe different levels of recreation. The Vienna Principles' definition (Kraker et al., 2016) focuses on traceability, others treat "reproducibility" and "replicability" either as interchangeable (Gentleman and Lang, 2007) or completely different terms (Goodman et al., 2016).

For the remainder of this work, we will use "reproduce" and "reproducibility" to say that a third party can run the original analysis using code and data provided by the author of a published work, and that this execution creates the same processing result (following a definition by Peng, 2009).

The referenced guides and rules indirectly define general challenges of reproducibility. Computational sciences, such as GEOBIA, face particular challenges. For example the uniqueness of data (can only be captured once) or processing environments (e.g. supercomputers) can make real replication of results impossible, so that trust in the applied methods must be established instead (Peng, 2011).

To achieve this trust, open sourcing of workflows and the underlying software is crucial. While benefits of free and open source software (FOSS) for business and security have been documented widely[5], a more important aspect of FOSS in science is the potential for evaluation and scientific collaboration. The licensing models[6] of FOSS allow to combine individual contributions of small functional parts into a bigger solution for a problem at hand. This modularity at the roots of many open source software projects is propagated by the *Unix philosophy* (Salus, 1994): Each programme should only provide a specific feature and excel at it. They must allow (technically and legally) maintenance and re-purposing by third parties.

A number of publications at GEOBIA conferences over the last years demonstrate the feasibility of a FOSS approach, for example using ILWIS and MultiSpec (Baldina and Grishchenko, 2014) or R and GRASS GIS (Van De Kerchove et al., 2014). Specific OBIA FOSS projects exist as well, for example InterIMAGE (Costa et al., 2010).

When implementing a complex workflow with FOSS, a large

*Corresponding author

Note: URLs in this document were last accessed July 1st 2016.

[1] https://www.fosteropenscience.eu/foster-taxonomy/open-science-definition

[2] See http://digital-scholarship.org/cwb/WhatIsOA.htm and https://open-access.net/DE-EN/information-on-open-access/history-of-the-open-access-movement

[3] https://ec.europa.eu/programmes/horizon2020/en/h2020-section/open-science-open-access

[4] http://ec.europa.eu/research/openscience/index.cfm?pg=open-science-cloud

[5] See for example https://opensource.org/advocacy/case_for_business.php and https://opensource.com/business/13/12/using-open-source-software

[6] For a quick introduction we recommend http://choosealicense.com/.

number of independent tools are utilised, both visible and invisible to the user. This hampers reproducibility because of compatibility conflicts between different software packages in different versions. In this study we present an approach to mitigate this problem by making the whole analysis including software in specific tested versions available through containerisation.

The implemented workflow detects destruction in a bitemporal image subset of a conflict area and is partly based on previous work (Knoth and Pebesma, 2014). Analysis of conflict damage is a use case where an open approach is specifically useful, because non-profit organisations face budget restrictions and, more importantly, because of the importance of transparency when using complex analysis techniques in politically sensitive environments.

The main contribution of this work is a fully reproducible and open workflow for geographic object-based image analysis (GEOBIA). It is based on mainstream IT containerisation technology and a collection of pieces of FOSS for image analysis and geographic information systems (GIS).

The following sections describe the image analysis workflow (Section 2), how it is implemented with FOSS, and how it is made reproducible using Docker (Section 3). Finally we discuss the solution and its challenges (Section 4) and conclude with a summary and outlook (Section 5).

2. EXAMPLE ANALYSIS - CONFLICT DAMAGE ASSESSMENT

2.1 Data

The data for our example analysis are two images of a village in Darfur, Sudan. They are available online as part of a blog post by the AAAS Geospatial Technologies Project[7]. The copyright holder DigitalGlobe granted permission to re-publish them as part of this work. The data consists of two preview pictures of remote sensing imagery showing the village Jonjona (located roughly at $13.686, 24.979$ (latitude, longitude) west of Al-Fashir) before (December 2004) and after (February 2007) reported attacks in the area (see Figure 1). They were downloaded from the website in $.jpg$ format, manually georeferenced, resampled to a spatial resolution of 0.5 metres (approximating the spatial resolution of current commercial very high resolution satellites), and saved as GeoTIFF files.

2.2 Analysis workflow

The general strategy implemented in this study is to segment the pre-conflict layer and analyse the resulting segments regarding their change values using information from the pre- and post-conflict temporal layers. The analysis workflow can be divided into three major steps: (i) feature extraction and segmentation, (ii) change analysis, and (iii) extraction of dwelling objects from changed objects. First, a principal component analysis (PCA) is applied to each file in order to compress the highly redundant spectral information of the three RGB bands to one dimension, the first principal component. The first principal components are then used as temporal layers of a bi-temporal data set. The image objects, as basic elements of the object-based analysis, are created using the watershed segmentation algorithm, which is based on the identification of local extrema (OTB Development Team, 2016).

[7] American Association for the Advancement of Science, http://www.aaas.org/page/appendix-darfur-sudan-and-chad-imagery-characteristics

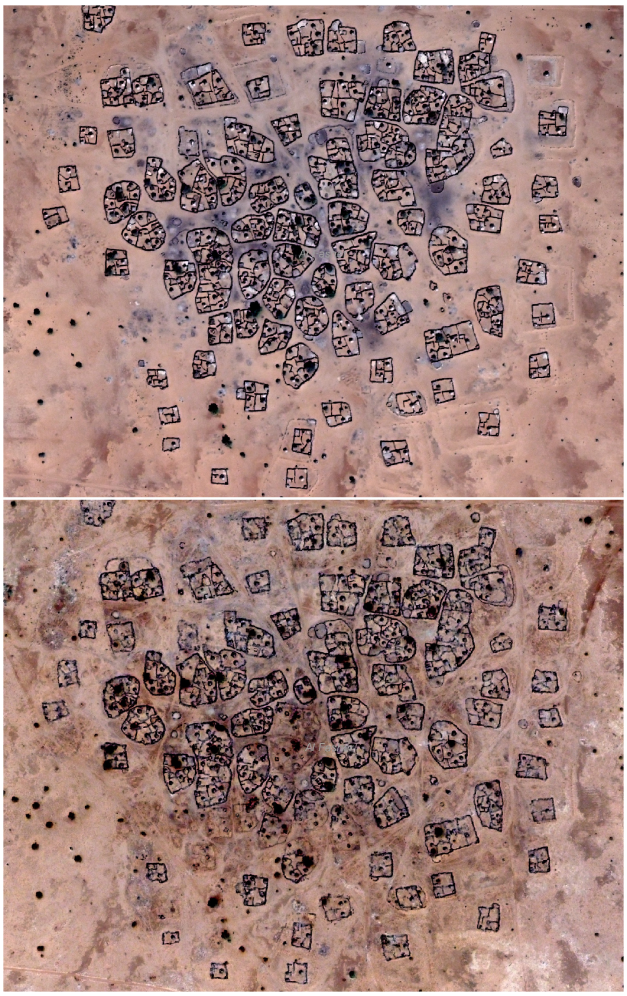

Figure 1. Example image of a village in Darfur before (December 2004, top) and after (February 2007, bottom) a reported attack. © 2016 DigitalGlobe

The segmentation is hampered by the specific structural properties of the objects of interest. Most of the dwellings have conical roofs. This results in a heterogeneous spectral response of the differently illuminated parts. Additionally, the buildings are often directly attached to fences or walls, so they sometimes poorly separate from the background. Earlier studies in similar areas have shown, that mathematical morphology can be used to eliminate such interfering features (Sulik and Edwards, 2010, Knoth and Pebesma, 2014). Therefore, a morphological closing operator precedes the segmentation to smooth out small and linear features. The objects resulting from the segmentation are analysed for structural differences between the two points in time. The basis for this change analysis is the difference in mean edge density per object. It is calculated after execution of an edge detection algorithm and used as the change attribute.

We apply two methods to extract the changed objects based on the change attribute: First, a fixed threshold, which can be tuned manually to best distinguish between changed and unchanged objects. Second, a k-means cluster analysis on all image objects, which does not need a predefined threshold. The latter is based on the assumption that when comparing the change of objects, disappeared dwellings differ significantly in the change of edge intensity from unchanged objects. Thus, they can be isolated in the cluster of highest change values using unsupervised clustering.

Besides the change analysis the objects are further investigated regarding their extent, their shape and their values in the pre-conflict morphological closing layer in relation to the unfiltered layer, i.e. the impact of the closing operator. This allows to better distinguish between changed dwelling structures and other, similarly changed objects (e.g. fences). The shape of objects is computed using the *Shape Index* (Lang and Blaschke, 2007). It measures how well an object approximates a circle, i.e. the more the shape differs from a circle, the higher the shape index value.

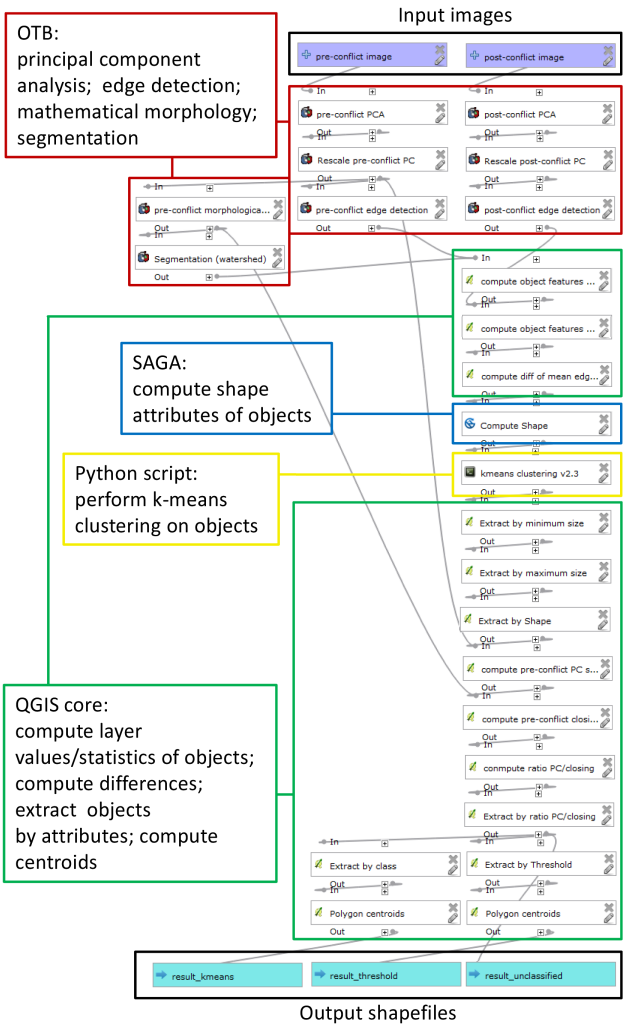

Figure 2. Overview of the analysis workflow in the graphical modeller and the role of the different software packages.

The workflow produces three results (see Figure 2). The first one is a point shapefile showing the centroids of dwelling objects detected as changed (disappeared/destructed) by the predefined threshold. The second output is another point shapefile indicating locations of dwellings belonging to the cluster of highest change as determined by k-means clustering (shown in Figure 3). The third product is a polygon shape file of all segments (changed and unchanged) where only the distinction between dwellings and other objects has been made. The change cluster (resulting from k-means clustering) as well as the computed change feature for each polygon are stored in the corresponding attribute table of this shapefile. This third result can be used to understand the two change detection results and to refine the analysis workflow (e.g. change number of clusters, adapt predefined thresholds etc.).

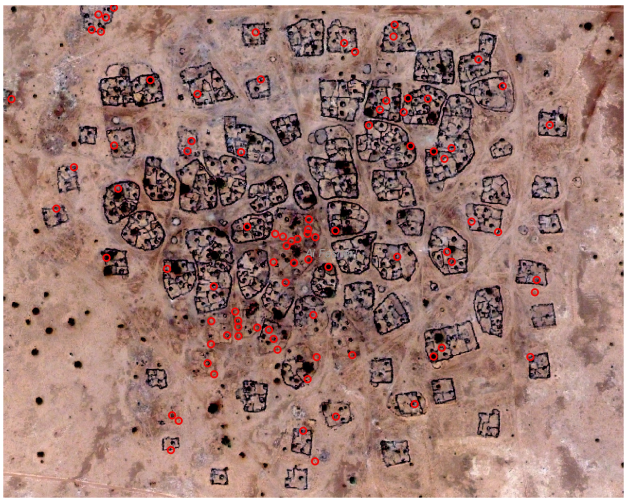

Figure 3. Post-conflict image with result of the destruction detection using k-means clustering of the change values (Image © 2016 DigitalGlobe).

3. PACKAGING OF THE GEOBIA WORKFLOW

3.1 Introduction

This section describes two software stacks for a FOSS-based implementation of the damage assessment workflow. They are evaluated to the extent that the respective implementations allow. The description is limited to the explicitly used pieces of software and leaves out the numerous underlying (system) libraries.

3.2 QGIS-based automated workflow

3.2.1 Developing the analysis model One implementation is based on the FOSS GIS *QGIS* (formerly known as Quantum-GIS, see QGIS Development Team, 2016). The *Processing Framework* included in QGIS (Graser and Oyala, 2015) provides access to native QGIS algorithms as well as a huge number of geoprocessing capabilities of third-party applications without requiring programming skills. In addition, user-created algorithms written in Python (Rossum, 1995), and subsequently the processing capabilities of any Python library, can be added.

The processing framework provides a graphical modeller for easy integration of the various algorithms into complex analysis models. These models can then be run as a whole on a selected set of input, e.g. layers in the QGIS desktop application.

Figure 2 gives an overview of the analysis steps and the involved software packages in the described workflow (Section 2.2) along with a screenshot of the corresponding modeller view. The applied tools comprise Orfeo Toolbox (OTB, see Inglada and Christophe, 2009) for image processing and segmentation tasks, native QGIS algorithms e.g. for computing object features based on the image layers, SAGA GIS (Conrad et al., 2015) for calculating shape attributes, and a user script performing a k-means algorithm using SciPy (Jones et al., 2016).

3.2.2 Workspace preparation The user workspace comprises the directories and files shown in Listing 1. They are stored in a specific directory structure so that the model executor (described in the next section) can identify them correctly. The contents of the workspace are

- a subdirectory `data` with the two original preview images and the georeferenced data files in TIFF format

- a Python script file, `model.py`, calling the actual model using the QGIS Python API[8]
- analogous to the Python user models and scripts directories, a `models` and a `scripts` directory containing respectively
  - a `.model`-file with the user model
  - a Python-file with a user algorithm

Listing 1. Workspace directory tree (documentation files not shown).

```
/workspace
|-- data
|   |-- COPYRIGHT
|   |-- Jonjona_after.jpg
|   |-- Jonjona_before.jpg
|   |-- jonjona_pos_conflict_proj.tif
|   `-- jonjona_pre_conflict_proj.tif
|-- model.py
|-- models
|   `-- conflict_damage_assessment.model
`-- scripts
    `-- kmeans_clustering_v2.3.py
```

The full workspace is available on GitHub[9].

3.2.3 Containerisation of workspace and runtime environment After the creation of a QGIS workflow with the QGIS graphical modeller, the next step is packaging all required parts of the analysis. We use a powerful tool for DevOps[10] called Docker[11]. It provides lightweight virtualisation to package an application and its dependencies, for example in cloud infrastructures.

In this study, we use a Docker image to encapsulate the GEOBIA workflow with a well-defined software environment. The image can be executed anywhere where a Docker host environment is running, including Linux, Windows, and OSX[12]. The image is build from a human- and machine-readable definition of the complete environment called `Dockerfile`. It allows a scripted definition of a Docker image, i.e. installation and configuration of contained software and files, and consequently a replication of a runtime environment. When an image is started and running it is called *container*. A container can be paused, stopped, and restarted, or be removed from the host. While not being intentioned for it, Docker is a means to ensure long term reproducibility of computational research, as demonstrated for example for R (Boettiger, 2015). A Docker image suffices to capture the data, software, and runtime environment in a well-defined manner and facilitates reproducibility.

In our specific case, the `Dockerfile` contains commands to install the required software, to copy the workspace into the container at the location `/workspace`, and to call a Bash[13] script to execute the actual workflow. The installation commands rely mostly on software packages from the Ubuntu[14] and UbuntuGIS-unstable[15] repositories. The exception is SAGA, which is installed from source in a specific version not available in the repositories to solve compatibility issues with QGIS[16].

The relevant parts of the main Bash script are shown in Listing 2. Omitted lines contain mostly logging commands. The main statements copy the models and script files from the workspace to the required QGIS locations and execute the Python script `model.py` with a virtual framebuffer using XVFB[17] because the container does not need a physical display. The copy statements use system environment variables which are shared between the different involved scripts, for example `$QGIS_MODELSCRIPT` resolves to the value `/workspace/model.py`.

Listing 2. Excerpt from the main bash script.

```
cp $QGIS_MODELFILE
    $QGIS_USER_MODELDIR/docker.model
cp $QGIS_SCRIPTFILE $QGIS_USER_SCRIPTDIR

xvfb-run python $QGIS_MODELSCRIPT
```

The Python script initiates and configures the QGIS application, for example library paths and logging. Then it creates variables holding the full paths to input and output objects and runs the actual model. Listing 3 shows an excerpt from the file. The `runalg` function's first argument, `"model:docker"`, loads the model in the file `docker.model` previously copied to the model directory by the Bash script.

Listing 3. Excerpt from the Python file for model execution: initiate QGIS and run the model.

```
app = QgsApplication([], True)
Processing.initialize()

processing.runalg("modeler:docker",
    inputimage_pre, inputimage_post,
    output_result_threshold,
    output_result_kmeans,
    output_result_unclassified)
```

3.2.4 Running the container The container can be executed on any local machine or server if Docker is installed. The image with the analysis is published on Docker Hub[18]. Only the first command shown in Listing 4 is required to run the container, because Docker downloads images automatically from Docker Hub.

Listing 4. Full reproduction commands: run the container from Docker Hub and extract the result.

```
docker run --name jonjona
  nuest/docker-qgis-model:knoth-nuest-geobia2016
docker cp jonjona:/workspace/results
    /tmp/jonjona_results
```

The second command copies the output of the workflow to a directory of the host machine. Its contents are shown in Listing 5. It contains a directory with a timestamp of the current execution, which contains three shapefiles—the actual model output. The shapefiles can now be inspected by the user or be processed further. Figure 3 shows a visualisation of the file `result_kmeans.shp`.

Listing 5. Workspace directory tree after execution (supplementary shapefile files, i.e. .dbf, .prj, .qpj, and .shx, and workspace files (see previous Listing 1 not shown).

```
/jonjona_results
|-- result
|   `-- 20160706-122126
|       |-- result_kmeans.shp
|       |-- result_threshold.shp
|       |-- result_unclassified.shp
```

[8] based on http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/intro.html#using-pyqgis-in-standalone-scripts

[9] https://github.com/nuest/docker-qgis-model

[10] http://radar.oreilly.com/2012/06/what-is-devops.html

[11] http://docker.io

[12] https://docs.docker.com/engine/installation/

[13] https://www.gnu.org/software/bash/

[14] http://archive.canonical.com/

[15] https://launchpad.net/ubuntugis/+archive/ubuntu/ubuntugis-unstable

[16] See http://hub.qgis.org/issues/13279 for details.

[17] https://en.wikipedia.org/wiki/Xvfb

[18] https://hub.docker.com/r/nuest/docker-qgis-model

3.3 InterIMAGE-based application package

InterIMAGE is another potential candidate for a FOSS-based OBIA workflow. It provides different segmentation algorithms[19] including the advanced and widely used multiresolution segmentation (Baatz and Schäpe, 2000) and operators for calculation of attributes like shape, texture or topological characteristics (Costa et al., 2010). A so-called *batch mode* feature[20], available since version 1.39, allows the automatic execution of InterIMAGE interpretation projects in the form of semantic networks. The networks store the classes and operators to be executed. Nüst and Knoth (2016) demonstrate running the user interface of the latest available Linux release (1.27) in a Docker container by sharing a local X11 socket.

However, several issues hinder the implementation of the presented use case (see Section 2). The software focuses on image interpretation and not all required algorithms for processing the image layers (e.g. the edge detection) are available in the basic package. More importantly, the latest available download for Linux is outdated[21]. We were not successful in compiling a later version of the source code for Linux as part of this work due to a lack of documentation and community support[22]. Currently only Linux is supported as the operating system inside a container, but support for multi-platform containers (most importantly Windows) is under development[23]. This limitation is not an issue for most FOSS projects because they are usually platform independent, but a straightforward execution of a InterIMAGE interpretation project within a container is not possible.

4. DISCUSSION

We successfully demonstrate packaging a complete GEOBIA workflow using FOSS. The created package is transferable between machines (different host operating systems as well as desktop and cloud platforms) and all tools are available free of charge. Our experiments show that because of the numerous involved tools in different versions and potential conflicts between them, containerisation is useful not only for reproducibility by third parties, but also for the original development of an FOSS-based analysis.

However, the approach still has shortcomings when it comes to the overarching goal of reproducible research. First, the presented solution is a one-off effort to containerise a specific workflow. It lacks a strict standardisation beyond the `docker run` command. Only experienced users can trace the command flow within the container. The currently used scripts are tailored to the actual use case, especially in the case of the file `model.py`. While most of the file is already generic, namely preparing the QGIS environment, logging, and clocking statements, the actual call to the algorithm is specific to the model. This part of the script should be split up so that only the relevant three to four lines of code must be provided by the workflow author.

Second, the `Dockerfile` installs automatically the latest available versions of software from the repositories. This proofed to be tricky during development of the use case with incompatibilities or missing features of libraries, resulting in a specific compatible version of one software to be installed manually. Software can be installed in specific versions, but creating these install statements manually is not user-friendly. While the used UbuntuGIS community repository simplifies installation tremendously, the ability to download, build, and install a software package from source within the `Dockerfile` was crucial to complete the use case at hand.

Third, the approach does not enforce best practices for reproducible research, such as versioning scripts, but could easily accommodate them, for example by putting a git repository inside the workspace directory (Ram, 2013).

Fourth, a general issue is the availability of open data. Especially in GEOBIA, where very high resolution imagery plays an important role in many analyses, the applied images are often not freely available. In these cases it is not possible to publish the data along with the analysis workflow and software.

Finally, the applied FOSS solutions, at the current stage, cannot compete with commercial software packages, such as eCognition, regarding usability and functionality in OBIA. The available functions of FOSS tools already provide a substantial set of algorithms and the analysis is created with a user-friendly interactive modeller in a Desktop environment. But the number of actual OBIA operations in the modeller is limited. They can only partially be re-created, e.g. by combination of other algorithms. However, since FOSS tools are easily extensible, the missing functionality can be contributed as new functions or independent tools. Therefore, we see a high potential for open-source software in OBIA, especially if a growing community of users can be established, who can become active contributors and lead the development of new algorithms.

5. CONCLUSION AND OUTLOOK

Docker containers and a combination of established free and open source GIS and image analysis software facilitate reproducible GEOBIA. We build and distribute a container to carry all required software and data in a transparent manner. This is a breakthrough for creating a transferable and executable package of a GEOBIA workflow. The presented analysis goes well beyond simple processing by successfully integrating a large set of tools into a complex multi-step analysis. The shortcomings discussed in the previous section are mostly related to usability. Therefore we see the following potential for future work.

With respect to standardisation, an open standard for packaging GEOBIA software and workflows would allow to follow similar approaches with different software stacks. This opens new possibilities for reviews of scientific work and collaboration between researchers.

The current solution is also mostly useful for users with software development and Linux experience. User interfaces that are delivered to a browser via HTTP can mitigate this limitation and provide a good user experience across platforms.

There is a need for documentation and ready-to-use templates for packaging as well as a user-friendly automation of packaging workflows, for example an “Export to Container”-button in the QGIS workflow modeller. A standardised format would also allow to create new services to store, share, and execute GEOBIA workflow packages in cloud infrastructures.

[19] http://wiki.dpi.inpe.br/doku.php?id=interimage:operators_documentation

[20] http://wiki.dpi.inpe.br/doku.php?id=interimage:batch_processing

[21] Version 1.27, see http://www.lvc.ele.puc-rio.br/projects/interimage/download

[22] https://groups.google.com/forum/#!topic/interimage/924t-uZrAMs

[23] See https://blog.docker.com/2016/04/docker-windows-server-tp5/ and https://www.docker.com/microsoft

With respect to interaction with the container, it would be possible to pass on parameters or external datasets into the container, for example via environment variables or mounting directories as volumes into the container. This way users can manipulate an analysis' parameters or quickly apply a complete workflow to their own data.

Regarding the outputs of the container, the presented solution lacks clear information about the result of the analysis, besides the log file and the created output files. More research on improving the result interpretation is required to enable machine-based output validation and better result visualisation for users.

But the tools are just one part of the deal: To reach high user-friendliness and adoption of an approach similar to this demonstration, we see a high demand in education of the current and next generation of OBIA users in programming and open source technologies. Although there are commonalities across all scientific disciplines, domain specific requirements demand targeted examples/course material, high-quality specialised FOSS, and best practises of common use cases. The challenge starts with a definition of reproducibility specifically for (GE)OBIA (Baker, 2016). This important next step needs an open discourse in the GEOBIA community, to which this work intends to contribute.

ACKNOWLEDGEMENTS

This research has has been conducted in the context of the Gradschool for Geoinformatics[24]. It has partly been supported by the project Opening Reproducible Research[25] funded by the German Research Foundation (DFG) under project number PE 1632/10-1.

REFERENCES

Baatz, M. and Schäpe, A., 2000. Multiresolution segmentation an optimization approach for high quality multi-scale image segmentation. In: J. Strobl, T. Blaschke and G. Griesebner (eds), *Angewandte Geographische Informations-Verarbeitung XII*, Wichmann, Karlsruhe, pp. 12–23.

Baker, M., 2016. Muddled meanings hamper efforts to fix reproducibility crisis. *Nature News*.

Baldina, E. A. and Grishchenko, M. Y., 2014. Object oriented analysis of multitemporal thermal infrared images. In: *Proceedings of the GEOBIA 2014: Geographic Object-Based Image Analysis*, Thessaloniki, Greece, pp. 415–418.

Boettiger, C., 2015. An introduction to docker for reproducible research, with examples from the r environment. *ACM SIGOPS Operating Systems Review* 49(1), pp. 71–79.

Conrad, O., Bechtel, B., Bock, M., Dietrich, H., Fischer, E., Gerlitz, L., Wehberg, J., Wichmann, V. and Böhner, J., 2015. System for Automated Geoscientific Analyses (SAGA) v. 2.1.4. *Geosci. Model Dev.* 8, pp. 1991–2007.

Costa, G., Feitosa, R., Fonseca, L., Oliveira, D., Ferreira, R. and Castejon, E., 2010. Knowledge-based interpretation of remote sensing data with the InterImage system: Major characteristics and recent developments. In: *Proceedings of the GEOBIA 2010: Geographic Object-Based Image Analysis*, Ghent, Belgium.

Gentleman, R. and Lang, D. T., 2007. Statistical analyses and reproducible research. *Journal of Computational and Graphical Statistics* 16(1), pp. 1–23.

[24] `http://www.uni-muenster.de/Geoinformatics/en/Studies/study_programs/PhD/`

[25] `http://o2r.info`

Goodman, S. N., Fanelli, D. and Ioannidis, J. P. A., 2016. What does research reproducibility mean? *Science Translational Medicine* 8(341), pp. 341ps12–341ps12.

Graser, A. and Oyala, V., 2015. Processing: A Python Framework for the Seamless Integration of Geoprocessing Tools in QGIS. *ISPRS International Journal of Geo-Information* 4(4), pp. 2219–2245.

Inglada, J. and Christophe, E., 2009. The orfeo toolbox remote sensing image processing software. In: *Proceedings of the 2009 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Cape Town, South Africa, Vol. 4, pp. 733–736.

Jones, E., Oliphant, T., Peterson, P. et al., 2016. SciPy: Open source scientific tools for Python. http://www.scipy.org/ (27 June 2016).

Knoth, C. and Pebesma, E., 2014. Detecting destruction in conflict areas in darfur. In: *Proceedings of the GEOBIA 2014: Geographic Object-Based Image Analysis*, Thessaloniki, Greece, pp. 165–168.

Kraker, P., Kaier, C., Gutounig, R., Vignoli, M., Dennerlein, S., Aspöck, E., Schmidt, N., Wandl-Vogt, E., Ferus, A., McNeill, G., Steinrisser-Allex, G., Dörler, D., Rieck, K., Heigl, F., imukovi, E. and Enkhbayar, A., 2016. The vienna principles: A vision for scholarly communication in the 21st century. *Zenodo*.

Lang, S. and Blaschke, T., 2007. *Landschaftsanalyse mit GIS*. Ulmer, Stuttgart, pp. 241–243.

Markowetz, F., 2015. Five selfish reasons to work reproducibly. *Genome Biology* 16, pp. 274.

Nosek, B. A., Alter, G., Banks, G. C., Borsboom, D., Bowman, S. D., Breckler, S. J., Buck, S., Chambers, C. D., Chin, G., Christensen, G., Contestabile, M., Dafoe, A., Eich, E., Freese, J., Glennerster, R., Goroff, D., Green, D. P., Hesse, B., Humphreys, M., Ishiyama, J., Karlan, D., Kraut, A., Lupia, A., Mabry, P., Madon, T., Malhotra, N., Mayo-Wilson, E., McNutt, M., Miguel, E., Paluck, E. L., Simonsohn, U., Soderberg, C., Spellman, B. A., Turitto, J., VandenBos, G., Vazire, S., Wagenmakers, E. J., Wilson, R. and Yarkoni, T., 2015. Promoting an open research culture. *Science* 348(6242), pp. 1422–1425.

Nüst, D. and Knoth, C., 2016. docker-interimage: Running the latest InterIMAGE linux release in a docker container with user interface. http://zenodo.org/record/55083 (06 July 2016).

OTB Development Team, 2016. The ORFEO Tool Box Software Guide. https://www.orfeo-toolbox.org//packages/OTBSoftwareGuide.pdf (27 June 2016).

Peng, R. D., 2009. Reproducible research and biostatistics. *Biostatistics* 10(3), pp. 405–408.

Peng, R. D., 2011. Reproducible research in computational science. *Science* 334(6060), pp. 1226–1227.

QGIS Development Team, 2016. QGIS Geographic Information System. http://qgis.osgeo.org (24 June 2016).

Ram, K., 2013. Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine* 8, pp. 7.

Rossum, G., 1995. Python reference manual, available at `http://www.python.org/`.

Salus, P., 1994. *A Quarter-Century of Unix*. Addison-Wesley, Boston, chapter 7 of part 2, p. 52.

Sandve, G. K., Nekrutenko, A., Taylor, J. and Hovig, E., 2013. Ten simple rules for reproducible computational research. *PLoS Computational Biology* 9(10), pp. e1003285.

Sulik, J. and Edwards, S., 2010. Feature extraction for darfur: geospatial applications in the documentation of human rights abuses. *International Journal of Remote Sensing* 31(10), pp. 2521–2533.

Van De Kerchove, R., Hanson, E. and Wolff, E., 2014. Comparing pixelbased and objectbased classification methodologies for mapping impervious surfaces in wallonia using orthoimagery and LIDAR data. In: *Proceedings of the GEOBIA 2014: Geographic Object-Based Image Analysis*, Thessaloniki, Greece, pp. 657–661.