Computers in Biology and Medicine 108 (2019) 49–56

Contents lists available at ScienceDirect

Computers in Biology and Medicine

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

DeepClas4Bio: Connecting bioimaging tools with deep learning frameworks for image classification

A. Inés*, C. Domínguez, J. Heras, E. Mata, V. Pascual

Department of Mathematics and Computer Science of University of La Rioja, Spain

ARTICLE INFO

Keywords:
Deep learning
Bioimaging
Image classification
Interoperability

ABSTRACT

Background and objective: Deep learning techniques have been successfully applied to tackle several image classification problems in bioimaging. However, the models created from deep learning frameworks cannot be easily accessed from bioimaging tools such as ImageJ or Icy; this means that life scientists are not able to take advantage of the results obtained with those models from their usual tools. In this paper, we aim to facilitate the interoperability of bioimaging tools with deep learning frameworks.

Methods: In this project, called DeepClas4Bio, we have developed an extensible API that provides a common access point for classification models of several deep learning frameworks. In addition, this API might be employed to compare deep learning models, and to extend the functionality of bioimaging programs by creating plugins.

Results: Using the DeepClas4Bio API, we have developed a metagenerator to easily create ImageJ plugins. In addition, we have implemented a Java application that allows users to compare several deep learning models in a simple way using the DeepClas4Bio API. Moreover, we present three examples where we show how to work with different models and frameworks included in the DeepClas4Bio API using several bioimaging tools — namely, ImageJ, Icy and ImagePy.

Conclusions: This project brings to the table benefits from several perspectives. Developers of deep learning models can disseminate those models using well-known tools widely employed by life-scientists. Developers of bioimaging programs can easily create plugins that use models from deep learning frameworks. Finally, users of bioimaging tools have access to powerful tools in a known environment for them.

1. Introduction

Image classification is a computer vision task that assigns a label to a given image — the label is chosen from a fixed set of categories. This is an instrumental tool in many life science problems, and deep learning techniques have become the state-of-the-art approach to deal with them. Just to name a few examples, deep learning has been applied for classifying breast cancer histology images [1], for classifying echocardiograms [2], for detecting and classifying nuclei in routine colon cancer histology images [3], for classifying lung nodules on computed tomography images [4], or for classifying skin cancer images [5]. In the aforementioned examples, and in general, whenever we want to use a deep learning model to solve a classification problem in bioimaging, we need to deal with two tasks: building the model, and, subsequently, integrating such a model into the workflow of the life scientists tools. Both steps present challenges for developers of bioimaging programs.

First of all, building a deep learning model requires some experience with tools and techniques that do not usually belong to the life scientists' toolbox. Moreover, we can find a large collection of different deep learning frameworks; each one having its own advantages and disadvantages [6,7]. Due to this fact, it might be difficult to choose the best framework to solve a concrete problem. Besides, sometimes, it is useful to try different alternatives and frameworks to tackle the same task.

Once the model is built, it remains the task of incorporating it into the workflow of life scientists' research. Deep learning models can be employed mainly inside the framework where they were built; and, in some cases, they can be exported to create, for instance, mobile apps [8]. However, there is not a general bridge to connect any bioimaging tool, like ImageJ [9] or Icy [10], with different deep learning frameworks. Since life scientists are used to work with bioimaging programs, integrating deep learning models on those bioimaging tools is a more

* Corresponding author.
E-mail addresses: adrian.ines@unirioja.es (A. Inés), cesar.dominguez@unirioja.es (C. Domínguez), jonathan.heras@unirioja.es (J. Heras), eloy.mata@unirioja.es (E. Mata), vico.pascual@unirioja.es (V. Pascual).

https://doi.org/10.1016/j.compbiomed.2019.03.026
Received 10 December 2018; Received in revised form 27 March 2019; Accepted 27 March 2019

sensible approach than creating special purpose interfaces — since life scientists already know the former.

In this work, we deal with the problem presented in the latter step; that is, connecting bioimaging processing tools with models built on different deep learning frameworks. Specifically, we focus on the classification problem. Recently, some works have tackled this problem; but, they have been focused on connecting a particular image processing tool with a concrete deep learning model [11] or framework [12]. On the contrary, the present project, called *DeepClas4Bio*, allows the connection of multiple bioimaging programs with several classification models from different deep learning frameworks. In order to carry out this task, we have developed an API that provides a common interface for some of the main deep learning frameworks, including Keras [13], Caffe [14], DeepLearning4J [15], PyTorch [16] and MxNet [17]; and can be easily extended to include others. This API might be employed with different aims, for instance comparing deep learning models, or extending the functionality of bioimaging programs (such as ImageJ [9] or Icy [10]) by creating plugins for those tools.

The DeepClas4Bio project brings to the table several benefits. We envision three potencial beneficiaries of DeepClas4Bio: developers of bioimaging tools, developers of deep learning models and users of bioimaging tools. From the perspective of developers of bioimaging tools, like ImageJ or Icy, they can create plugins that employ a wide variety of models provided by different deep learning frameworks; moreover, and thanks to the common interface of DeepClas4Bio, changing the framework, or model, to explore different alternatives is a simple task. From the perspective of developers of deep learning models, they can easily integrate their models in the DeepClas4Bio project, facilitating their use thanks to the graphical interface of bioimaging tools; and hence, overcoming the lack of a by-default easy-to-use interface in deep learning frameworks. In addition, since deep learning models can be distributed as plugins of bioimaging tools, their dissemination is easier. Finally, from the perspective of users of bioimaging tools, they can apply deep learning techniques to their problems in an easy way.

2. Methods

The structure of the DeepClas4Bio project can be split into two sides, see Fig. 1. On the one side, there are image classification models provided by the deep learning frameworks; and, on the other side, there are bioimaging tools. In order to integrate these two parts, we have defined the DeepClas4Bio API, which provides a common interface for inference using classification models of deep learning frameworks. The definition of such an API is instrumental to allow the interaction with different models and frameworks, since each tool has its own

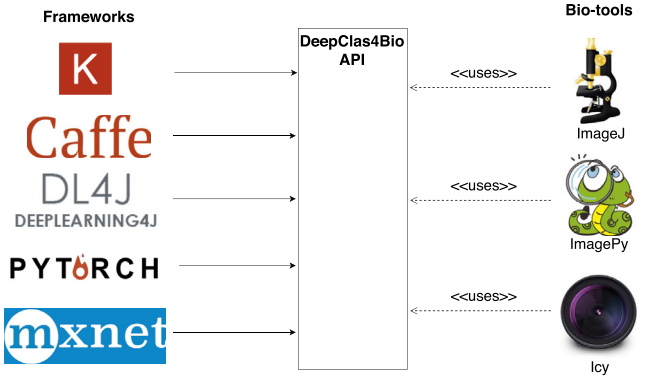

Fig. 1. DeepClas4Bio is an API that sits in between deep learning frameworks and pretrained models, and common bioimaging tools. It enables users to employ those models to make predictions on new images with minimal code from model developers and no code from the end user.

peculiarities — for instance, the programming language, the way of storing models and weights, the kind of layers, the representation of images, or the way of returning the results. Therefore, we defined a common interface to facilitate, not only its use by third-party tools, but also the incorporation of new models and frameworks in the future. The API can be called from bioimaging tools by creating plugins that extend the functionality of those programs, and it also could be employed to create standalone tools as we will illustrate in Section 4.

The DeepClas4Bio API has been developed in Python. We have chosen this language because the majority of deep learning frameworks offer support for it; and, in addition, several libraries that are helpful for deep learning (for instance, the scientific computing package Numpy [18] or the library for computer vision OpenCV [19]) are available in this language. Moreover, Python can be easily connected with other programming languages like Java.

The developed API has five public methods. The first method, called listFrameworks, allows users to know the frameworks included in the API. Currently, the API gives support for Keras, Caffe, DeepLearning4J, PyTorch and MxNet. The second method, called listModels, gives the available deep models for a particular framework; for instance, the networks VGG [20], AlexNet [21], ResNet [22] and GoogleNet [23], trained for the ImageNet challenge [24], are available in Keras. The third method, called predict, serves to classify an image given a model and a framework; and a similar method, called predictBatch, is also available to predict the category of a batch of images. The last, but not least, method, called evaluate, allows users to compare the quality of several models with respect to a dataset using different measures (accuracy, rank5, precision, recall, F1-score, Jaccard Index, Matthews Correlation and AUROC are available). All the methods of the API share the same output format, a JSON file — a very widespread format that is easy to use.

The DeepClas4Bio API is not a static library; but, it has been designed to easily include new models and frameworks. This is an important issue in a discipline like deep learning where new models appear almost in a daily-basis, and also new frameworks appear from time to time [25,26]. In particular, the design of the API is based on the factory design pattern [27], that allows us to include new functionalities without modifying the previous work. Using this pattern, we can create the requested models in a simple way. In addition, when adding a new model or framework to the API, it will not be necessary to modify the existing code. A detailed explanation of the procedure to extend the API is provided in the next section.

The API can be installed via PyPi [28] on the final users' computers. In order to use this version of the API, the final users should also install the corresponding deep learning frameworks and libraries locally — the DeepClas4Bio webpage provides the steps to carry out this process. Hence, if developers create an application using this local version of the API, they must also provide its installation procedure for the final users. Since in some cases the installation of deep learning frameworks might be cumbersome for final users, we also provide an example of a web service template, implemented using the Flask framework [29], that shows developers how to create web applications that use the API. Such a web application reduces the burden of installing different libraries in the final users' computers, and can be particularized for each concrete problem. This requires that developers deploy a server and deal with all the web-services issues, like security, authentication, data transfer or web-interface.

In order to incorporate the prediction capabilities of the API to the bioimaging tools, developers of those tools can implement new plugins — an extensible system provided by most bioimaging tools. Namely, the plugin's developer only needs to:

1. Invoke the predict method of the API with the selected framework and model.
2. Process the result, the JSON file, returned by the API and show the result.

Hence, the final goal of the aforementioned procedure is to use the API to classify an image — that is, given an image, the plugin will allow the user to invoke a model and obtain a class for that image. Such a procedure can be applied not only for the classification of an image but also to other tasks such as obtaining predictions of different models and comparing them, obtaining the prediction of a particular region of an image, classifying a batch of images instead of a single image, and, in general, solving any task that involves an image classification process.

The images are transferred from bioimaging tools to the API using their paths. We have decided to use this option since each deep learning framework works with a different representation of images, and therefore is easier to open the image in the necessary format than developing several transformers.

3. Architecture of the DeepClas4Bio API

The DeepClas4Bio API has been designed using several classes that abstract the procedures to classify images using models, and to evaluate several deep learning models using different measures.

Independently of the deep learning framework and model used, the procedure to classify an image, or batch of images, using a deep learning model can be summarized as follows: (1) load the model; (2) preprocess the image or batch of images; (3) make the prediction of an image or batch of images; and (4) postprocess the output produced in (3). Steps (1), (2) and (4) depend on the model; and step (3) can be seen as a framework operation, since all the models of a framework can be evaluated with the same instruction. Therefore, we have modelled steps (1), (2) and (4) using an abstract class Model (see Fig. 2), that is particularized for each concrete model; and step (3) by a Predictor class that is particularized for each framework. So, when the user invokes the predict method of the DeepClas4Bio with an image, framework and model, DeepClas4Bio internally constructs the corresponding Predictor and Model instances, preprocesses the image with the preprocessor of the Model instance, obtains a prediction with the predict method of the framework, and postprocesses that result using the postprocessor of the model. In order to implement this architecture, we have employed the dependency injection principle and the factory pattern [27].

This architecture can be easily extended to include new models, by adding new classes that extend the Model class, and new frameworks, by implementing new classes that extend the Predictor class.

The procedure to evaluate several deep learning models for a dataset using different measures follows the next steps: (1) load the dataset, (2) select the models to evaluate, (3) select the measures used to evaluate the models, and (4) evaluate the models selected in (2) for the dataset loaded in (1) using the measures selected in (3). Step (1) depends on how the dataset is organized. Steps (2) and (3) depend on the models and the measures selected. Finally, Step (4) depends on the evaluator built with the dataset, the predictors and the measures. Therefore, we have modelled Step (1) using an interface IReadDataset, see Fig. 3, that is particularized for each concrete way to load a dataset; Steps (2) and (3) with the corresponding Predictor class and the measures (given as functions); and Step (4) by a Evaluator class.

4. Results

In this section, we present several examples and applications that use the DeepClas4Bio API. In particular, we introduce a metagenerator that allows developers to easily create ImageJ plugins; a Java application that allows users to compare several deep learning models in a simple way using the DeepClas4Bio API; and, finally, we present three case studies where we show how to work with different models and frameworks included in the API using different bioimaging tools. All the tools and plugins presented in this section can be downloaded from the project webpage (see the availability section).

4.1. A metagenerator for ImageJ plugins

As we mentioned at the end of Section 2, the plugins that use the DeepClas4Bio API always follow a common pattern. Therefore, developers of bioimaging tools interested in creating plugins using the API will always follow the same steps, and the structure and code of those plugins will be very similar — they will mainly differ on the name of the model and framework employed. Since this task might be repetitive, it is interesting to automate it. Also, developers of such bio-plugins could be not expert developers. Therefore, an automatic tool to create these plugins could help them. To achieve such a goal, we have developed a metagenerator of ImageJ plugins. This metagenerator is a Java application that allows users to create specific plugins for the deep models included in the DeepClas4Bio API.

The procedure to use this metagenerator is straightforward. First of all, the metagenerator shows the frameworks and the models available in the API; then, the developers of bioimaging tools choose the framework and model; and, finally, the metagenerator generates automatically the code of the corresponding ImageJ plugin, i.e. the Java code and the corresponding pom file. Hence, developers could modify this base code to adapt it to their purposes and generate new plugins in a simple and fast way. Currently, this metagenerator is only available to generate ImageJ plugins but, similar tools for other image processing programs can be easily created.

4.2. A java application to evaluate models

There are a lot of different frameworks and models for classifying images. Due to this fact, selecting the most suitable option for a

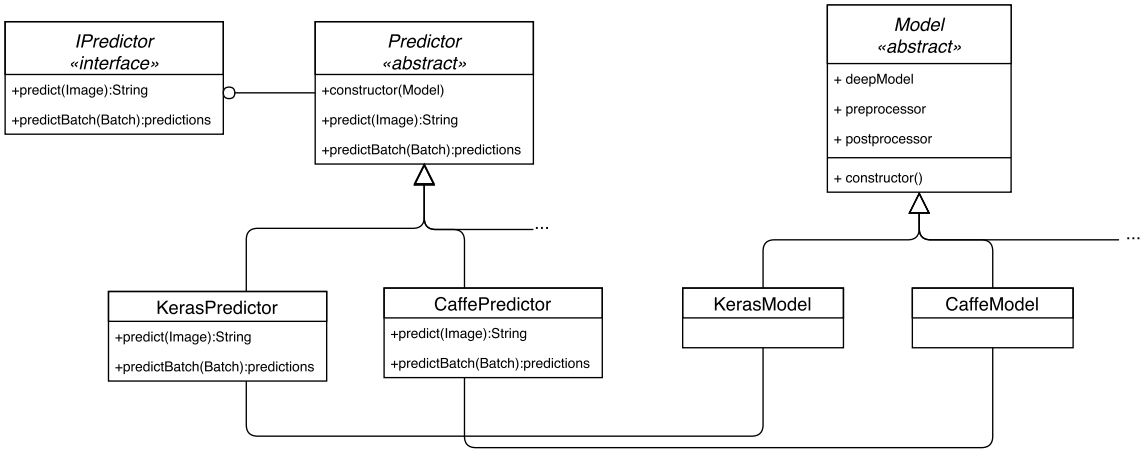

Fig. 2. Class Diagram of the DeepClas4Bio API for including models and frameworks.

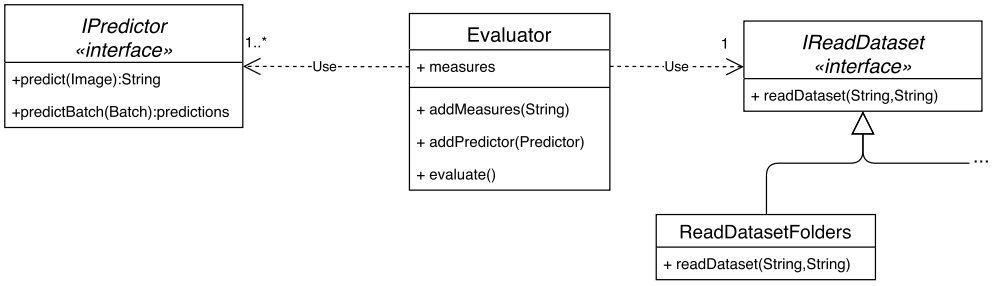

Fig. 3. Class Diagram of the DeepClas4Bio API for evaluating models.

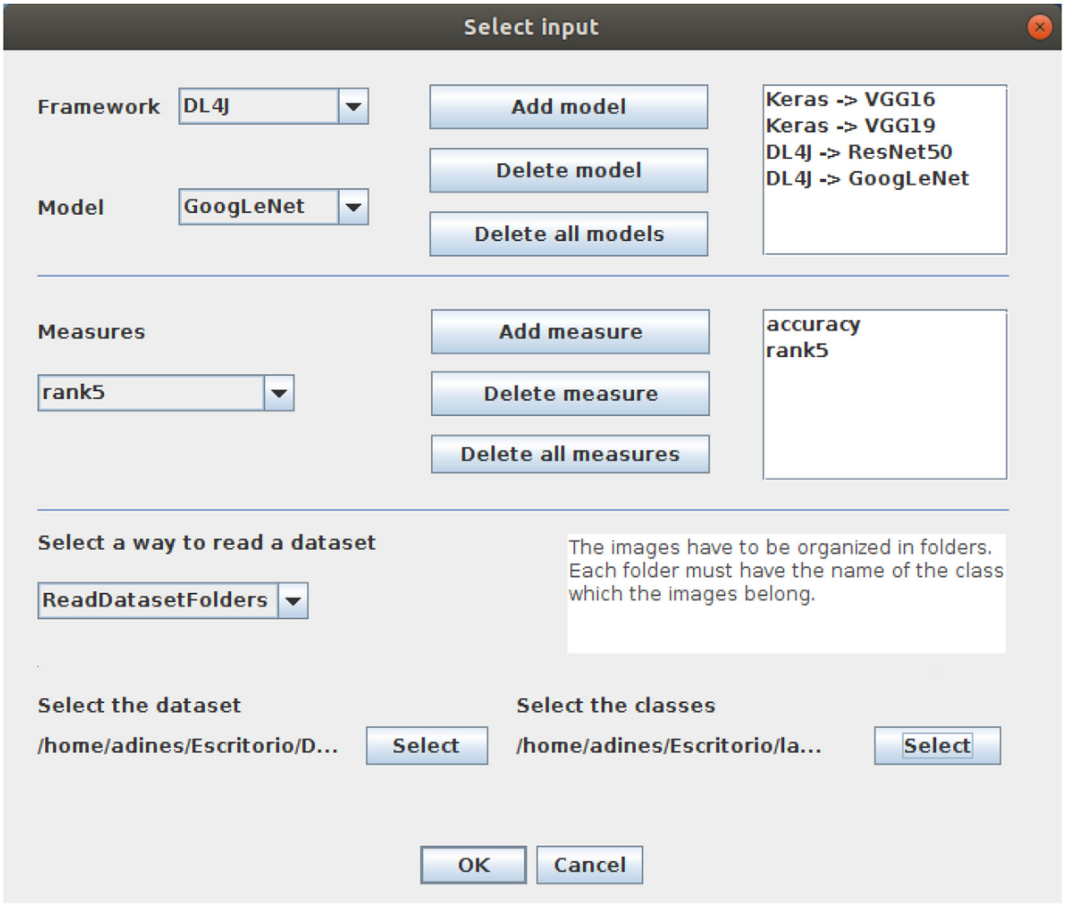

Fig. 4. The DeepClas4Bio API enables the comparison of several models on different frameworks, using a developed Java application, in instances where the classification labels are shared across models and a ground truth dataset is provided. The user does not need to write any code.

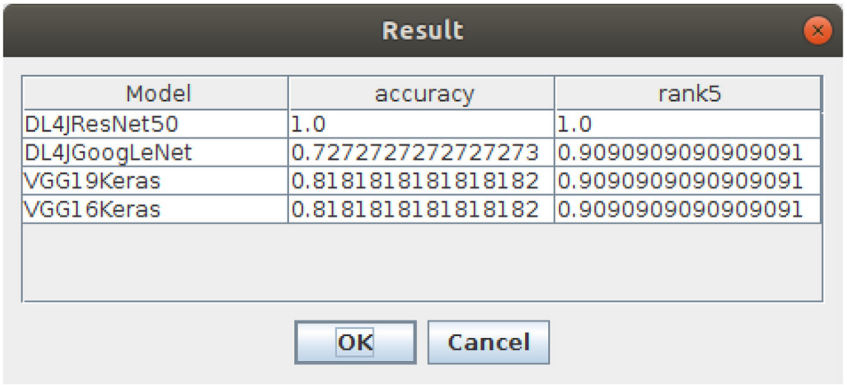

Fig. 5. Various metrics computed on a ground truth dataset for various models compared using the Java application of Fig. 4.

particular problem might be a challenge. For this reason, we have developed a Java application that allows users to compare different models in a simple way using the DeepClas4Bio API. This application is an example of the use of the API, that shows the different possibilities offered by such an API to work with different frameworks and models. This application facilitates the comparison of different models for a specific problem, however it is the responsibility of the final developer to choose the model to be used in each specific problem.

In the Java application, see Fig. 4, the users can select the models to compare, the dataset to test and the measures to use. The models that the users can select are the models included in the DeepClas4Bio API. Also, we have included the main measures to evaluate models in the DeepClas4Bio API, including, accuracy, rank5, precision and recall among others. In addition, the users can select the most suitable option to load their datasets.

Once the users have selected all the necessary options to compare the models, the application connects with the DeepClas4Bio API to obtain the results. These results are presented to the users in a table as shown in Fig. 5.

4.3. Creating plugins for bioimaging tools

We consider three different approaches to work with different classification models, and we also connect our API with three different bioimaging programs.

4.3.1. Case study 1: comparing ImageNet models

The first example here shows how several models, that are already available in different deep learning frameworks, can be invoked from the DeepClas4Bio API and then easily accessed from ImageJ. Namely, we consider the problem of classifying an image in one of the 1000 categories of the ImageNet challenge [24] using some of the models that have won such a challenge along the last years. In particular, we employ the pretrained models VGG16 [20], ResNet [22] and GoogLeNet [23] available in the *DeepLearning4J*, *MxNet* and *PyTorch* frameworks to create an ImageJ plugin that allows us to try all these models.

Let us briefly describe the different tools involved in this case study. On the bioimaging side, we have ImageJ [9], an open-source digital image processing Java program that has been designed with an open architecture that provides extensibility via plugins and recordable macros; its simple-to-use interface and the wide variety of features supplied by the plugins make ImageJ a very popular tool in the scientific and educational fields. ImageJ is built on top of SciJava [30], a project that aims for the collaboration among different tools by reusing several Java libraries for scientific computing. On the deep learning side, we have *DeepLearning4J*, a Java framework for deep learning; *MxNet*, a deep learning framework that gives support for several programing languages, namely Python, Scala, R, Julia and Perl; and *PyTorch*, a Python deep learning framework based on the Torch library [31]. By combining all these tools, we have created an ImageJ plugin to compare the results of several models in a simple way. First of all, the users loads an image; then, the plugin connects to the API, installed locally in the user computer, to obtain, using the listFrameworks and listModels methods of the API, the available frameworks and models to prepare the dialog that will be shown to the user. Finally, the plugin calls the API, using the predict method, to classify the image with the selected models, and shows the result to the user, see Fig. 6.

4.3.2. Case study 2: classifying melanoma images

In this second example, we show how to work with a fine-tuned model and save it in the API to classify different types of melanomas in ImagePy. For this task, we have used the ResNet model and a subset of the ISIC melanoma dataset [32]. Fine-tuning [33] is a transfer learning technique [34] employed to take advantage of a model that was trained for a problem in order to solve a different task. In particular, fine tuning consists in performing “network surgery” by modifying the structure of the network so we can re-train parts of it. This strategy has been applied successfully in several projects [35,36]; and, it is specially useful when only a limited amount of data is available.

For this task, we have employed Keras, an open source neural network library written in Python. This framework is a high-level API for neural networks, capable of running on top of TensorFlow, CNTK, or Theano. The deep learning framework Keras has a set of pretrained models that were trained with the ImageNet dataset, including ResNet. In addition, it provides all the necessary tools to conduct fine-tuning. Once the model for the ISIC melanoma dataset was built (the code to fine-tune the model can be found in the project webpage), we included it in our framework, see Appendix A.

Once the model is included in the API, it can be used. To facilitate its usage, we have built an ImagePy plugin, see Fig. 7. ImagePy [37] is an image processing program very similar to ImageJ. Since ImagePy is written in Python, it should be easy to integrate it directly with the deep learning frameworks — at least easier than integrating the deep learning frameworks with tools written in a different language. However, such a direct integration will require to know the particularities of each framework; on the contrary, the integration using DeepClas4Bio is simpler thanks to the common provided interface.

4.3.3. Case study 3: a plugin for the classification of gastrointestinal diseases

In this last example, we show that is also possible to work with a batch of images instead of classifying a single image. In particular, we have trained the *ResNet* model in *PyTorch* with a gastrointestinal disease dataset — the Kvasir dataset [38,39]. All the necessary code to train this model is available in the project webpage.

For this example we have used PyTorch as deep learning framework. PyTorch is an open source deep learning framework developed by Facebook's artificial-intelligence research group. This deep learning framework is based on Torch and written in Python and C++. Once that such a model was built, we included it in the API, see Appendix A. After including the model in the API, we developed an Icy plugin. Icy [10] is an open-source Java image processing program developed by the *Unité d'analyse d'images quantitative - CNRS URA 2582 - Institut Pasteur* that can also be extended via Java plugins. The workflow of this Icy plugin is straightforward. When this plugin starts, the users have to select the folder with the images they want to classify. Then, the plugin connects with the API to classify the images using the predictBatch method, see Appendix A. Finally, the plugin shows the information in a table, see Fig. 8.

As we have shown in this section, it is quite straightforward to add new models to the DeepClas4Bio API and integrate them into different bioimaging tools.

5. Discussion

In the DeepClas4Bio project, we try to improve the interoperability between deep learning frameworks and bioimaging processing programs. There are other projects that connect bioimaging tools with both deep learning and machine learning libraries. In this section, we comment them and compare them with our approach.

There are several projects that connect a concrete bioimaging tool with a particular deep learning or machine learning library. The *ImageJ - TensorFlow* plugin [12] connects ImageJ with the deep learning framework TensorFlow [40]; specifically, this project translates between ImageJ images and TensorFlow tensors allowing ImageJ users to apply the most important deep learning techniques and models included in TensorFlow in an easy way. Similarly, CellProfiler [41] has been connected with both Tensorflow and Caffe [42]. Another project that integrates machine learning techniques in ImageJ is *Trainable Weka Segmentation* [43], a plugin that combines a collection of machine learning algorithms with a set of selected image features to produce pixel-based

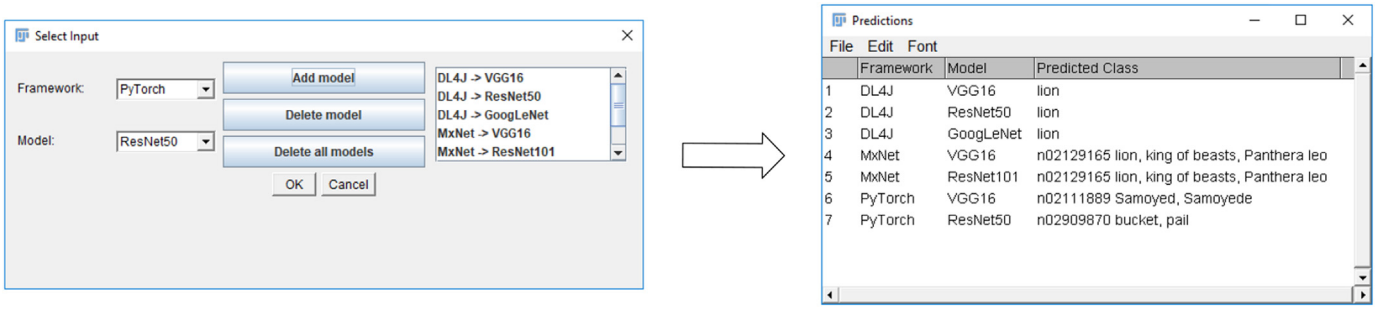

Fig. 6. ImageJ plugin comparing the prediction of several pretrained models.

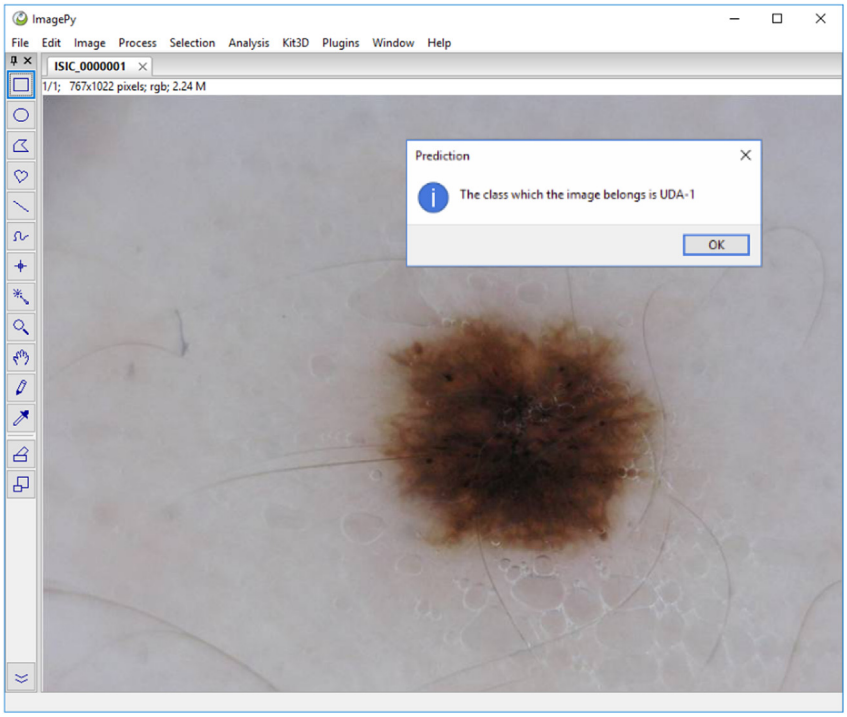

Fig. 7. Example ImagePy plugin leveraging the API's predict method to classify a melanoma image based on a model that was fine-tuned in Keras. While code needs to be written and run to generate the fine-tuned model, the actual plugin itself requires only a small amount of python code thanks to the API, and running the plugin to make predictions on new images requires writing no code.

segmentations thanks to Weka [44]. *Rapid Learning* [45] is a similar project that integrates machine learning techniques into Icy; in this case, the machine learning engine used is RapidMiner [46], a powerful data mining tool. KNIME [47], a data analytics platform, has included some deep learning frameworks and models in its platform.

The aforementioned connections have been already exploited to create useful plugins. For example, *Microscope Focus Quality* [11], is an ImageJ plugin that predicts an absolute measure of image focus on a single image in isolation without any user-specified parameters. This plugin uses a pre-trained deep neural network for the prediction of the image focus employing the *ImageJ - TensorFlow* project. Other example is *Measure Image Focus* [48], a CellProfiler plugin in the same line.

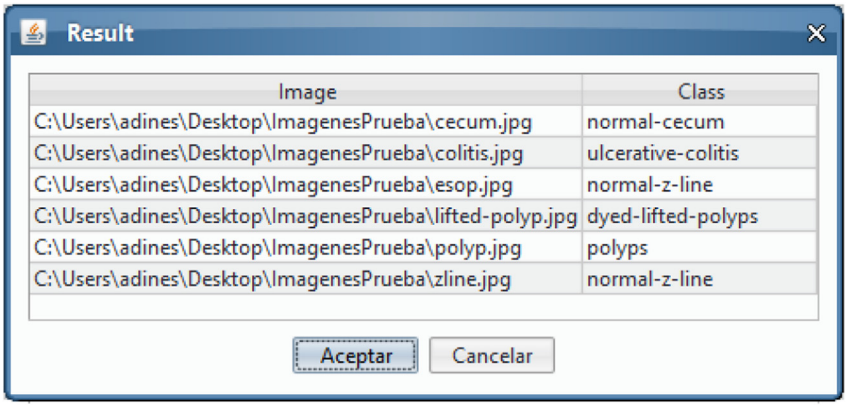

Fig. 8. Example Icy plugin that classifies a batch of images of gastrointestinal diseases using a PyTorch model trained from scratch. Java code is written to add the model to the API, and to implement the plugin's predict call to the API, but the end user does not need to run any code to make new predictions.

Our approach to integrate deep learning techniques with bioimaging tools is more general; namely, we do not restrict our project to a particular bioimaging tool or deep learning library, but we have created an API that can include models from any deep learning library, and it is intended to be integrated in any bioimaging tool. Our API provides an intermediate connection between bioimaging tools and the deep learning libraries by hiding the low-level details of each particular framework. Another difference is that concrete integrations, such as *ImageJ - TensorFlow*, give access to all the features of the deep learning library, provided that the developer has the required experience; on the contrary, we focus on classification models already trained on those libraries. This facilitates the integration of models for the developers who do not have experience using the deep learning libraries.

A challenge that we have faced during the development of DeepClas4Bio was the variability of deep learning frameworks, and the difficulties of sharing models and information across them. There are several active projects that are focused on transforming models among deep learning libraries. Some of these solutions are collected in a project called *Deep learning model converter* [49]. One of the problems to connect different deep learning frameworks is that each one of them uses its own specification and proprietary representation of models. In the *Open Neural Network Exchange (ONNX)* project [50], a common representation of the computation graph has been created. The idea of the ONNX project is to build the model with this representation and then use the framework that better adapts to the users' goals to execute it. Also, in this project different converters have been created to transform this representation to the each framework particular representation. In the same line is the dnn module of OpenCV [51]. In that module, a representation close to Caffe's representation has been developed, and some frameworks and models have been converted. This OpenCV module could be connected to bioimaging programs using bridges like IJ-OpenCV [52], that connects ImageJ with OpenCV. The main problem with such an approach is that some frameworks do not implement all types of layers of deep learning models; so, certain models cannot be converted to all the frameworks. In addition, if new tools and frameworks appear in the future, it would be necessary to create new converters. The advantage of our solution is that the developers can work natively with the format of each framework, avoiding conversion problems, and allowing the integration of new frameworks and models in the API without waiting for the development of converters.

In summary, DeepClas4Bio offers several benefits. Namely, DeepClas4Bio groups together the main deep learning frameworks giving an abstract layer to use them thanks to a common API; and, it is straightforward to extend the API to include new frameworks and models. In addition, the development of plugins that use the DeepClas4Bio API is simple and facilitates the use of deep learning models by life-scientists.

6. Conclusions

DeepClas4Bio is a free and open-source project that allows the collaboration of bioimaging tools with image classification models developed in deep learning frameworks. This project has been successfully employed to construct several plugins in different bioimaging tools. Thanks to the DeepClas4Bio API, and the methodology presented in this paper to integrate it in the workflow of life scientists' research, the development efforts can be greatly reduced when creating new tools for bioimaging that use deep learning classification models.

Currently, the deep learning models included in the API are focused on the classification task. Including other kind of deep learning models for localization, detection and semantic segmentation requires further research and will be tackled in the future. For including those models, the main infrastructure of the project can remain unchanged, and we only need to extend our class diagram. Also, as further work, would like to provide life scientists with the necessary tools to easily train deep models from scratch using our framework.

Availability

The DeepClas4Bio project is a free open source project. This project can be downloaded from https://github.com/adines/DeepClas4Bio where the interested reader can find the documentation and all the examples presented in the paper.

Conflicts of interest

None declared.

Acknowledgments

This work was partially supported by Ministerio de Economía, Industria y Competitividad, project MTM2017-88804-P; Agencia de Desarrollo Económico de La Rioja, project 2017-I-IDD-00018; and FPU Grant 16/06903 of the Spanish Ministerio de Educación y Ciencia.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.compbiomed.2019.03.026.

References

[1] T. Araújo, G. Aresta, E. Castro, J. Rouco, P. Aguiar, C. Eloy, et al., Classification of breast cancer histology images using convolutional neural networks, PLoS One 12 (6) (2017).
[2] A. Madani, R. Arnaout, M. Mofrad, R. Arnaout, Fast and accurate view classification of echocardiograms using deep learning, npj Digit. Med. 1 (6) (2018).
[3] K. Sirinukunwattana, S.E.A. Raza, Y.W. Tsang, D.R.J. Snead, I.A. Cree, N.M. Rajpoot, Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images, IEEE Trans. Med. Imaging 35 (5) (2016) 1196–1206.
[4] K.L. Hua, C.H. Hsu, S.C. Hidayati, W.H. Cheng, Y.J. Chen, Computer-aided classification of lung nodules on computed tomography images via deep learning technique, OncoTargets Ther. 8 (2015) 2015–2022.
[5] A. Esteva, B. Kuprel, R.A. Novoa, J. Ko, S.M. Swetter, H.M. Blau, S. Thrun, Dermatologist-level classification of skin cancer with deep neural networks, Nature 542 (2017) 115–118.
[6] V. Kovalev, A. Kalinovsky, S. Kovalev, Deep learning with theano, Torch, Caffe, Tensorflow, and Deeplearning4J: which one is the best in speed and accuracy? Proceedings of Pattern Recognition and Information Processing (PRIP 2016), Publishing Center of BSU, Minsk, 2016.
[7] S. Bahrampour, N. Ramakrishnan, L. Schott, M. Shah, Comparative study of Caffe, neon, theano, and Torch for deep learning, Proceedings of International Conference on Learning Representations (ICLR 2016), Workshop Track, 2016.
[8] Google, TensorFlow Lite, (2018) [Online]. Available https://www.tensorflow.org/lite/.
[9] C.T. Rueden, et al., ImageJ2: ImageJ for the next generation of scientific image data, BMC Bioinf. 18 (2017) 529.
[10] F. de Chaumont, S. Dallongeville, N. Chenouard, N. Hervé, S. Pop, T. Provoost, V. Meas-Yedid, P. Pankajakshan, T. Lecomte, Y. Le Montagner, Icy: an open bioimage informatics platform for extended reproducible research, Nat. Methods 9 (7) (2012) 690–696.
[11] S.J. Yang, M. Berndl, D.M. Ando, M. Barch, A. Narayanaswamy, E. chrisitansen, et al., Assessing microscope image focus quality with deep learning, BMC Bioinf. 19 (1) (2018) 77.
[12] Google Brain team, ImageJ/TensorFlow Integration Library Plugin, (2018) [Online]. Available https://imagej.net/TensorFlow.
[13] F. Chollet, Keras: the Python Deep Learning Library, (2015) [Online]. Available https://keras.io.
[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: convolutional architecture for fast feature embedding, Proceedings of the 22nd ACM International Conference on Multimedia, ACM, 2014.
[15] Eclipse Deeplearning4j Development Team, Deeplearning4j: Open-Source, Distributed Deep Learning for the Jvm, (2018) [Online]. Available https://deeplearning4j.org/.
[16] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in PyTorch, Proceedings of Neural Information Processing Systems (NIPS 2017), Workshop, 2017.
[17] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, MxNet: a flexible and efficient machine learning library for heterogeneous distributed systems, Proceedings of Neural Information Processing Systems (NIPS 2015), Workshop on Machine Learning Systems, 2015.

[18] T.E. Oliphant, A Guide to Numpy, CreateSpace Independent Publishing Platform, 2015.
[19] A. Kaehler, G. Bradski, Learning OpenCV 3, O'Reilly Media, 2015.
[20] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, Proceedings of International Conference on Learning Representation, ICLR 2015, 2015.
[21] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances In Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.
[22] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, 2016 IEEE Conference On Computer Vision And Pattern Recognition (CVPR 2016), 2016, pp. 770–778.
[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, 2015 IEEE Conference On Computer Vision And Pattern Recognition, CVPR 2015), 2015, pp. 1–9.
[24] J. Deng, W. Dong, R. Socher, L.J. Li, K. Li, L. Fei-Fei, ImageNet: a large-scale hierarchical image database, 2009 IEEE Conference On Computer Vision And Pattern Recognition, 2009, pp. 248–255.
[25] ModelZoo, Model Zoo: Discover Open Source Deep Learning Code and Pretrained Models, (2018) [Online]. Available https://modelzoo.co/.
[26] Y. Jia, Caffe — Model Zoo, (2018) [Online]. Available http://caffe.berkeleyvision.org/model_zoo.html.
[27] K. Arnout, B. Meyer, Pattern componentization: the factory example, Innov. Syst. Softw. Eng. 2 (2) (2006) 65–79.
[28] Python Software Foundation, Python Package Index, (2019) [Online]. Available https://pypi.org/.
[29] M. Grinberg, Flask Web Development: Developing Web Applications with Python, first ed., O'Reilly Media, Inc, 2014.
[30] C. Rueden, J. Schindelin, M. Hiner, K. Eliceiri, SciJava Common [Software], (2016) http://scijava.org/.
[31] R. Collobert, S. Bengio, J. Mariéthoz, "Torch: a modular machine learning software library," IDIAP, Tech. Rep. (2002) Idiap-RR-46-2002.
[32] D. Gutman, N.C.F. Codella, M.E. Celebi, B. Helba, M.A. Marchetti, N.K. Mishra, A. Halpern, Skin lesion analysis toward melanoma detection: a challenge at the international symposium on biomedical imaging (ISBI) 2016, hosted by the international skin imaging collaboration (ISIC), Conference On Computer Vision And Pattern Recognition, CVPR 2017, 2017.
[33] A. Rosebrock, Deep Learning for Computer Vision with Python, PyImageSearch, 2017.
[34] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.
[35] A. Kumar, J. Kim, D. Lyndon, M. Fulham, D. Feng, An ensemble of fine-tuned convolutional neural networks for medical image classification, IEEE J. Biomed. Health Int. 21 (1) (2017) 31–40.
[36] G.J. Scott, M.R. England, W.A. Starms, R.A. Marcum, C.H. Davis, Training deep convolutional neural networks for land–cover classification of high-resolution imagery, IEEE Geosci. Remote Sens. Lett. 14 (4) (2017) 549–553.
[37] W. Anliang, Y. Xiaolong, W. Zhijun, ImagePy: an open-source, Python-based and platform-independent software package for bioimage analysis, Bioinformatics 34 (18) (2018) 3238–3240 [Online]. Available https://dx.doi.org/10.1093/bioinformatics/bty313.
[38] K. Pogorelov, K.R. Randel, C. Griwodz, S.L. Eskeland, T. de Lange, D. Johansen, et al., KVASIR: a multi-class image dataset for computer aided gastrointestinal disease detection, Proceedings Of the 8th ACM on Multimedia Systems Conference, Ser. MMSys'17, ACM, New York, NY, USA, 2017, pp. 164–169.
[39] A. Asperti, C. Mastronardo, The effectiveness of data augmentation for detection of gastrointestinal diseases from endoscopical images, Proceedings of the 5th International Conference, BIOIMAGING 2018, 2018.
[40] Google, TensorFlow: an Open-Source Software Library for Machine Intelligence, (2018) [Online]. Available https://www.tensorflow.org.
[41] T.R. Jones, I.H. Kang, D.B. Wheeler, R.A. Lindquist, A. Papallo, D.M. Sabatini, P. Golland, A.E. Carpenter, Cellprofiler analyst: data exploration and analysis software for complex image-based screens, BMC Bioinf. 9 (1) (2008) 482.
[42] A. Carpenter, Cellprofiler 3.0 Release: Faster, Better, and 3d, (2018) [Online]. Available https://blog.cellprofiler.org/2017/10/16/cellprofiler-3-0-release-faster-better-and-3d/.
[43] I. Arganda-Carreras, et al., Trainable Weka Segmentation Plugin, (2018) [Online]. Available https://imagej.net/Trainable_Weka_Segmentation.
[44] Machine Learning Group at the University of Waikato, Weka 3: Data Mining Software in Java, (2018) [Online]. Available https://www.cs.waikato.ac.nz/ml/weka/.
[45] W. Ouyang, Rapid Learning - Icy Plugin, (2013) [Online]. Available http://icy.bioimageanalysis.org/plugin/Rapid_Learning.
[46] O. Rittho, R. Klinkenberg, S. Fischer, I. Mierswa, S. Felske, Yale: yet another learning environment, Tagungsband der GI-Workshop, Woche Lernen- Lehren - Wissen Adaptivitat, 2001, pp. 84–92.
[47] M.R. Berthold, N. Cebron, F. Dill, T.R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, B. Wiswedel, KNIME: the konstanz information miner, in: C. Preisach, H. Burkhardt, L. Schmidt-Thieme, R. Decker (Eds.), Data Analysis, Machine Learning And Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 319–326.
[48] S.J. Yang, M. Berndl, D.M. Ando, M. Barch, A. Narayanaswamy, E. chrisitansen, et al., Measure Image Focus - A CellProfiler Plugin, (2018) [Online]. Available https://github.com/CellProfiler/CellProfiler-plugins/wiki/Measure-Image-Focus.
[49] S. Yuan, Deep Learning Model Convertors, (2018) [Online]. Available https://github.com/ysh329/deep-learning-model-convertor.
[50] Microsoft, Facebook open source, AWS, ONNX: Open Neural Network Exchange, (2018) [Online]. Available http://onnx.ai/.
[51] R. Alexander, Deep Learning, Now in Opencv, (2017) [Online]. Available https://habr.com/company/intel/blog/333612/.
[52] C. Domínguez, J. Heras, V. Pascual, IJ-OpenCV: combining ImageJ and OpenCV for processing images in biomedicine, Comput. Biol. Med. 84 (2017) 189–194.