DeepImageJ

**General Description**

DeepImageJ allows any user without previous programming knowledge to apply deep learning models to their image processing tasks. Moreover, it is a user-friendly tool that contributes to the spread of deep learning models, breaking the existing technological barriers between computer science and applied science.

The software stores the conditions and parameters needed to make inference on any given image in a file along with the actual model. This file is retrieved every time the model is called, allowing the user to apply it with no knowledge about the characteristics of the neural network architecture.

**Software**

Limitations

This software can be used to apply only deep learning models stored in the SavedModel format of TensorFlow. These models must have been developed with TensorFlow versions <=1.13.1.

The plugin only works for models with one input tensor and one output tensor. Both the input and the output must be images, so tasks such as prediction or classification are excluded.

In addition to this, the software is limited to 2D images. This tool only admits images with one slice and one time frame (although the number of channels is not limited).

Pre-Requirements

The software provided here consists of two plugins for **Fiji**, a general purpose image-processing and image-analysis package. Fiji has a public domain licence; it runs on several platforms: Unix, Linux, Windows, and Mac OS X. It doesn't take more than a couple of minutes to install.

Instructions

The software consists of two Fiji plugins. The developer plugin, which is used to create a stable, robust and self-explanatory version of the TensorFlow model. The output of the developer plugin can be applied with the other plugin, the end-user plugin, to make inference on an image.

Developer Plugin

This plugin is used to define the define the different parameters and conditions needed to run a TensorFlow model on any given image.

The plugin loads the wanted TensorFlow SavedModel and creates a folder which contains the model, an xml configuration file, an pair of images consisting of a sample image and the result of applying the model on that image and two text files which contain the ImageJ macros (if needed) for pre-processing and post-processing.

Load a TensorFlow SavedModel on Fiji

**First select a valid model**

In order to get started, a TensorFlow SavedModel has to be loaded. For that a valid TensorFlow SavedModel has to be introduced in the text box.

A valid TensorFlow SavedModel consists on a folder which contains a file called “saved\_model.pb”, with the architecture of the model, and another folder called “variables” with the weights of the model.

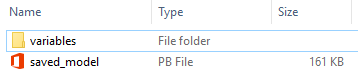


Figure 1 Files needed for a TensorFlow SavedModel

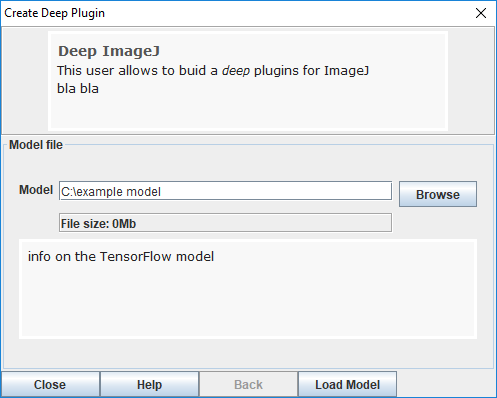


Figure 2 Introduce the path to the wanted model

Once a valid model folder has been introduced the plugin will try to open and load the model whenever we press “Next”.

**Open and load the model**

In order to load and apply the model two “tags” defined at the moment the model was saved are needed. These are the MODEL TAG and the SIGNATURE DEFINITION.

These keywords have to be defined by the programmer of the network whenever they save the model. However, there exist some default MODEL TAGs and SIGNATURE DEFINITIONs defined by TensorFlow which the plugin will use to try to load the model. If it is possible to load the model with them, they will appear as default in their corresponding text box, if it is not these text boxes will appear blank allowing the user to introduce their particular MODEL TAG and SIGNATURE DEFINITION.

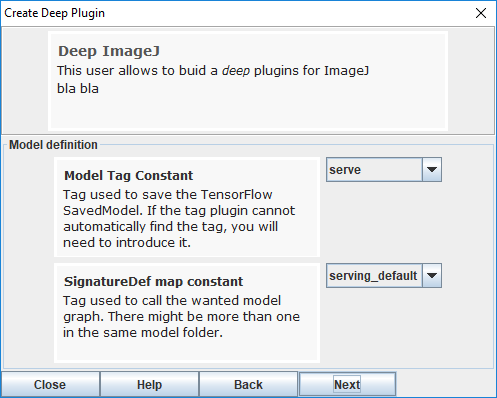


Figure 3 In this case the plugin was able to find the needed tags to open the model

**Define the tensor dimensions and conditions required to run the model**

Once the model is loaded the plugin retrieves the dimensions of the input and output tensors for the model. The developer of the network must assign to every dimension what does it represent, either batch size, width, height or number of channels.

The programmer must also define if the tensor introduced in the model has fixed dimensions on the X and Y axes or not. In the case that the size of the tensor needs to be fixed the developer will need to introduce the size in X and Y of the tensor, and if not, they will introduce the minimum size (if possible) for which the model works or just the size with which the model is normally applied.

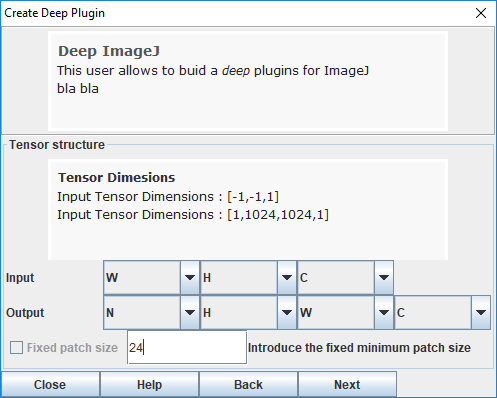


Figure 4 In this screen the dimensions of the tensor are identified. The size of the patch is also identified. In this case the patch can have an arbitrary size as long as it is a multiple of 24x24 pixels.

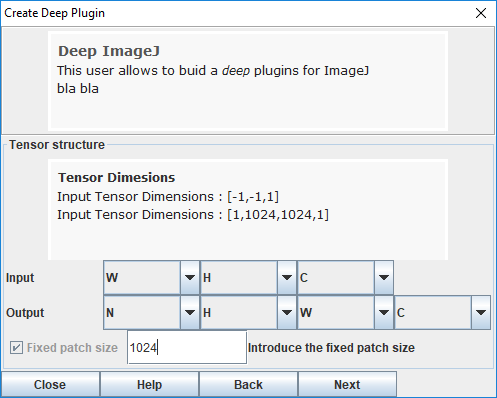


Figure 5 For this case the size of the patch to run the model has to be 1024x1024.

Due to memory limitations and in most of the cases, the images will need to be divided in patches in the X and Y dimensions to make inference more affordable.

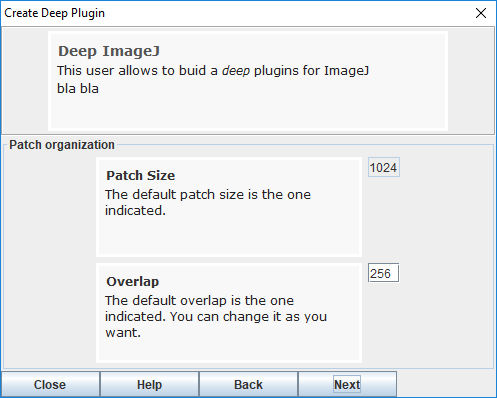


Figure 6 Now define the patch size again if it can be modified (not in this case) and the overlap.

Therefore, the plugin makes the user decide the size in X and Y of the patch in which the original image will be divided and that will be passed to the network, one at a time.

The parameters to be defined are the patch size and the overlap.

The patch size can be fixed if the developer decided so in the previous screen. If not, the default patch size (which can be modified) will be the smallest multiple of the integer introduced over 200. The developer will be able to modify this number as long as it is a multiple of the number he introduced in the previous screen.

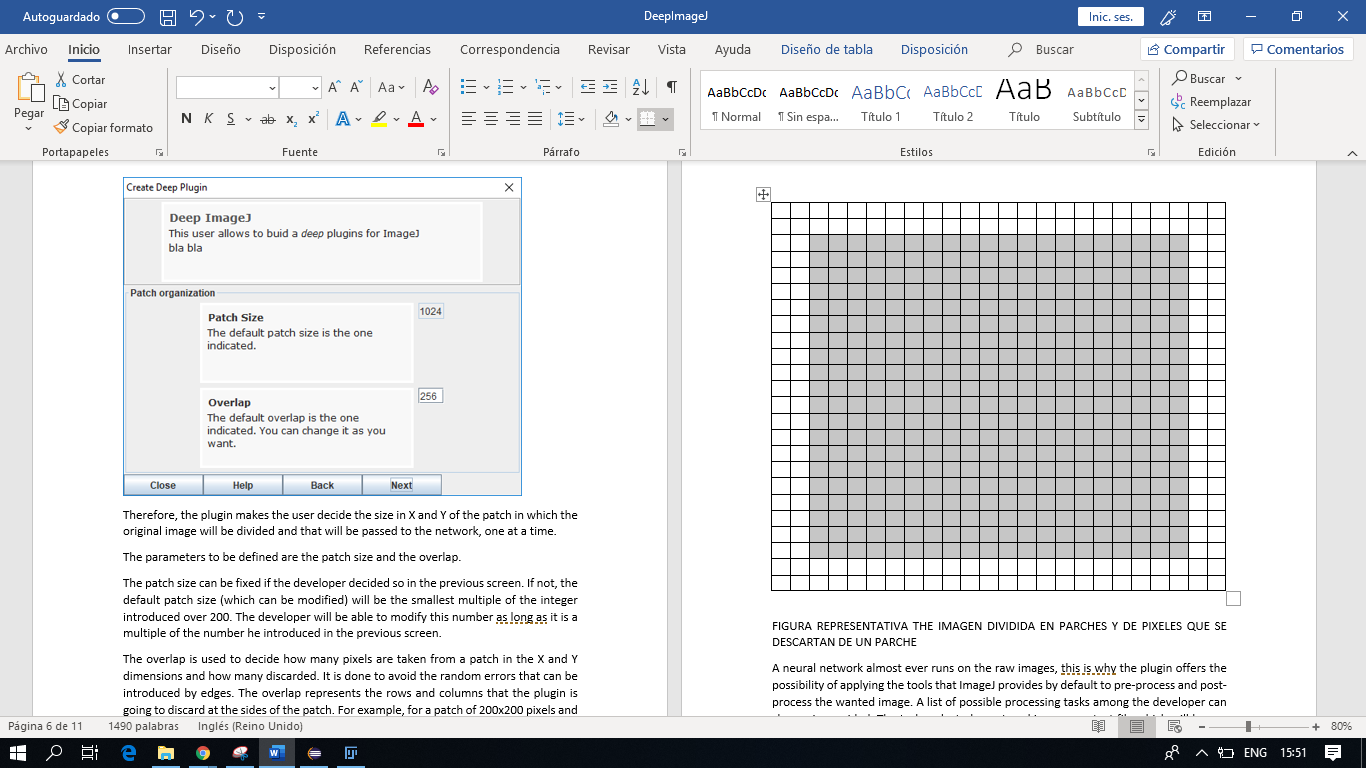
The overlap is used to decide how many pixels are taken from a patch in the X and Y dimensions and how many discarded. It is done to avoid the random errors that can be introduced by edges. The overlap represents the rows and columns that the plugin is going to discard at the sides of the patch. For example, for a patch of 200x200 pixels and overlap of 10 pixels, the patch that the plugin will use to reconstruct the resulting image will be the portion of 180x180 pixels of the middle.

Figure 7 Example of what overlap means. This grid represents a 24x24 patch. If the overlap for it was 2 pixels, only the highlighted pixels forming a 20x20 image would be considered.

**Define the processing**

A neural network almost ever runs on the raw images, this is why the plugin offers the possibility of applying the tools that ImageJ provides by default to pre-process and post-process the wanted image. A list of possible processing tasks among the developer can choose is provided. The tasks selected are stored in a macro text file which will be run on the image when necessary.

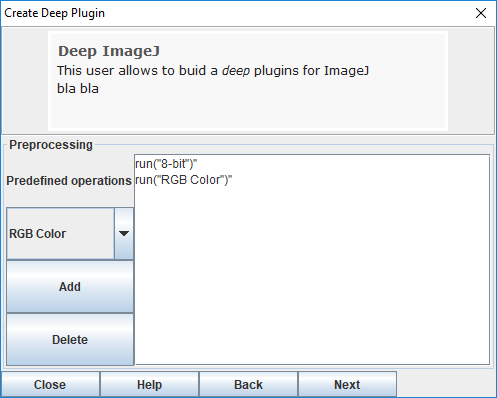


Figure 8 Example of preprocessing.

**Test and save the model**

Once all these parameters and conditions have been defined for the correct functioning of the network, the model is ready to be saved. But first the plugin requires the model to be tested with an example image in order to test that the model actually can be applied.

The model can be tested by pressing the button “Test”. Then the plugin will apply the corresponding pre-processing macros first, then it will divide the image in patches and run the model on each of these patches. With the resulting images, the plugin will reconstruct the output image and will apply the corresponding post-processing macros to eventually show the final result.

If the model works and it can produce an output image, the “Save” button will be enabled and the user will be allowed to save the “model”.

The output saved by the plugin consists on a folder (as the model folder) containing the model (the architecture and the weights), a configuration file “config.xml” with the parameters to run the network robustly, the example image used by the user to test the model, the result image produced by that test and two text files called “preprocessing.txt” and “postprocessing.txt” which contain the macros needed to process the image.

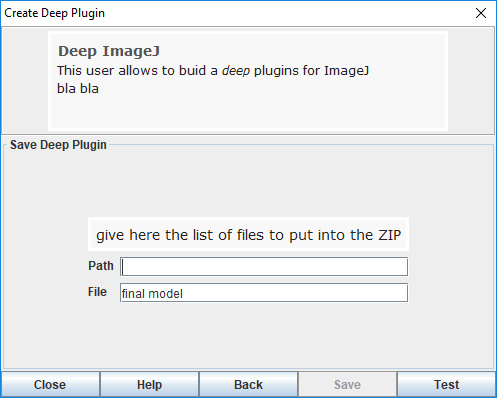


Figure 9 If the model has not been tested, saving it is not allowed.

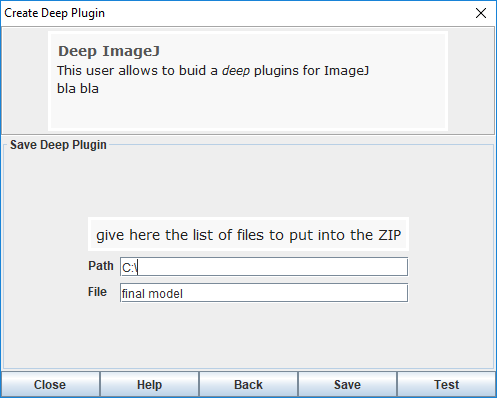


Figure 10 Once the model has been tested, it can be saved.

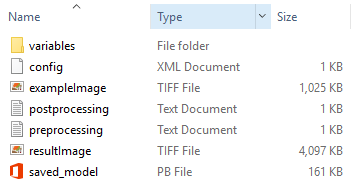


Figure 11 Files generated saving the model. These files are necessary to run the model in the user plugin.

USERS PLUGIN

This part of the software allows the end user to make inference with deep learning models on the wanted images without needing to have any background in deep learning.

This plugin can load and apply deep learning models produced by the other plugin without having to introduce any parameter.

When this plugin is started, it looks for a folder called “models” inside the Fiji directory, and if it does not exist, it creates it.

If the folder exists, the plugin looks for valid models inside it. Valid models are folders created by the other plugin (DeepImageJ) which contain at least a TensorFlow SavedModel, a configuration xml, a pre-processing text file and a post-processing text file.

If the plugin finds any valid model, it shows them in form of a list to the user. In order to apply any of these models, the user has to select it and load it (“Load Model”), then the button “Apply” will be enabled for the user to apply the model to any wanted image.

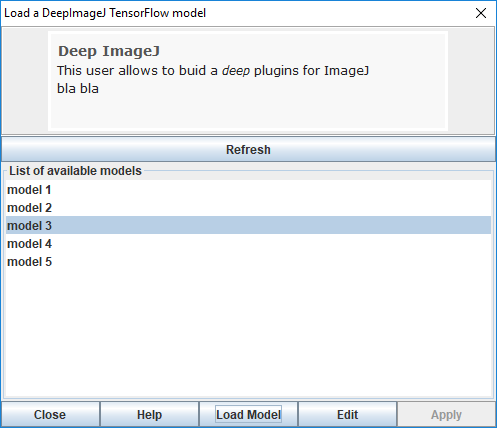


Figure 12 List of available models. No model can be applied because no model has been loaed.

If the user wants to use another model, they will only need to select “Change Model” and the current model will be unloaded for the user to select another one and the button “Apply” will be disable again.

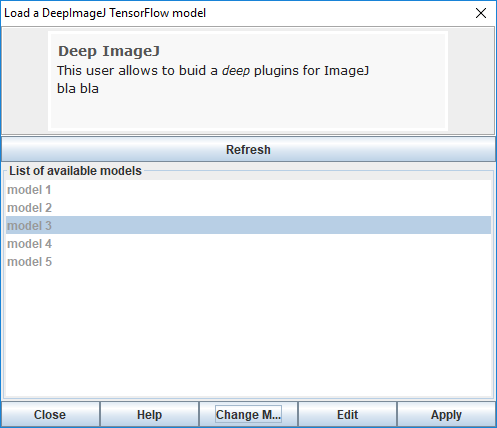


Figure 13 After a model has been loaded, it can be applied, edited or changed.

**The button “Edit”**

When the model is applied after loading it, the network is run on the image using all the default parameters defined by the actual developer of the deep learning model. However, more experienced users might want to change some parameters due to memory or performance issues.

If the user selects “Edit”, a new dialog box will appear where they will have the possibility of changing the size of the patch into which the original image is divided, or the overlap used between those patches used to reconstruct the result image.

The patch size will only be possible to modify if the developer of the network decided so. And in the case that it is possible, the patch size will have to be a multiple of a factor stated by the developer.

The overlap can be modified to any number, but it might have impact on the quality of the results obtained, therefore it is recommended to use the parameters defined by the developer.

It is also possible to decide whether pre-processing or post-processing are applied or not although the results of the network might be altered. Even in the case that the pre-processing includes a task crucial for the network, such as changing the number of channels, the model might be unable to run.

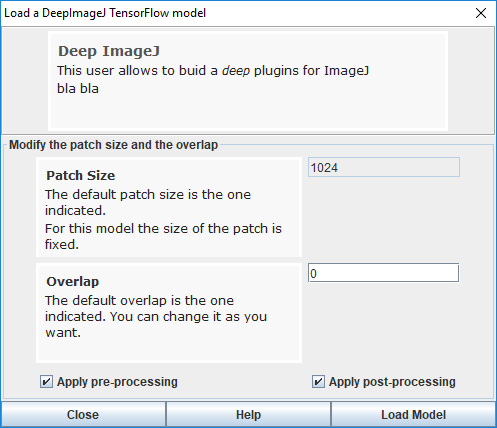


Figure 14 In the edit mode, the application of the model can be modified slightly.