ADAMS

Advanced **D**ata mining **A**nd **M**achine learning **S**ystem

Module: adams-imaging

Peter Reutemann

June 10, 2013

DRAFT

©2009-2013

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Except where otherwise noted, this work is licensed under
http://creativecommons.org/licenses/by-sa/3.0/

Contents

1 Java Advanced Imaging **7**

2 ImageJ **9**
2.1 Flow 9
2.2 Plugins 11

3 ImageMagick **13**

4 Object conversion **15**

5 Interaction **17**

6 WEKA output **21**

7 Miscellaneous actors **23**

Bibliography **25**

List of Figures

1.1 JAI flow for blurring images stored in a directory. 8
1.2 The original image. 8
1.3 The blurred image. 8

2.1 ImageJ flow for turning images stored in a directory into greyscale ones. 10
2.2 The original image. 10
2.3 The greyscale image. 10

3.1 ImageMagick flow for processing (resizing) a single image. 14
3.2 ImageMagick commands to resizing. 14
3.3 The original image. 14
3.4 The resized image. 14

5.1 Flow for generating ARFF file from user-labelled pixels. 18
5.2 User interface for labelling pixels, displaying some pixels labelled already. 18
5.3 Example dataset generated using the PixelSelector. 19

6.1 Generating an ARFF file using ImageJ. 21
6.2 The ImageJ generated ARFF file. 22

Chapter 1

Java Advanced Imaging

Java Advanced Imaging (JAI) is an API to provide a simple, high-level programming model which allows developers to create their own image manipulation routines[1].

There are five JAI actors available:

- `source.JAICreateImage` – for creating an empty image to be forwarded.
- `transformer.JAIReader` – for reading any image file that JAI supports[2] and forwarding a `BufferedImageContainer` object.
- `transformer.JAITransformer` – performs a transformation using an existing JAI transformer class on the incoming image and outputs another image again.
- `transformer.JAIFlattener` – turns a `BufferedImageContainer` into an `weka.core.Instance` object to be used in WEKA. The attaced meta-data in form of a report can be added to the output object as well.
- `sink.JAIWriter` – for writing a `BufferedImageContainer` to a file format that JAI supports. If the image type cannot be determined based on the extension, you can also specify which type to generate.

Figure 1.1 shows a flow[3] for reading images, blurring them using a gaussian blur transformer and displaying them side-by-side. Figures 1.2 and 1.3 show original and blurred image.

[1] `http://en.wikipedia.org/wiki/Java_Advanced_Imaging`
[2] `http://java.sun.com/products/java-media/jai/iio.html`
[3] adams-imaging-gaussian_blur.flow

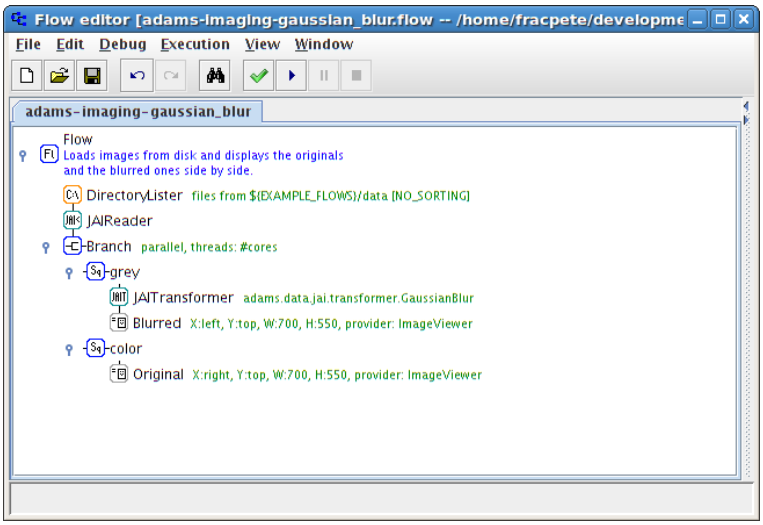

Figure 1.1: JAI flow for blurring images stored in a directory.

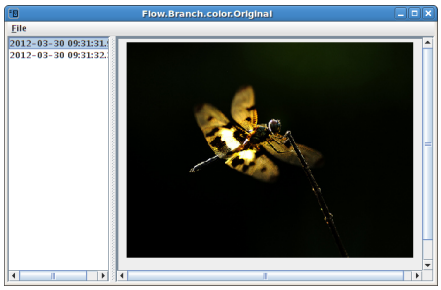

Figure 1.2: The original image.

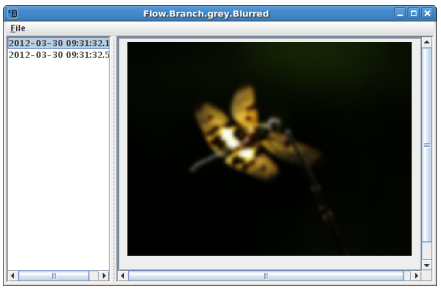

Figure 1.3: The blurred image.

Chapter 2

ImageJ

ImageJ is a public domain software suite written in Java (using AWT, opposed to Swing which ADAMS uses) for image processing, developed at National Institutes of Health ([3]).

2.1 Flow

There are four ImageJ actors available:

- `transformer.ImageJReader` – for reading any image file that JAI supports[1] and forwarding an `ImagePlusContainer` object.
- `transformer.ImageJTransformer` – performs a transformation using an existing ImageJ transformer class on the incoming image and outputs another image again. ImageJ plugin filters, commands and pre-recorded macros can be used to perform transformations.
- `transformer.ImageJFlattener` – turns an `ImagePlusContainer` into an `weka.core.Instance` object to be used in WEKA. The attaced meta-data in form of a report can be added to the output object as well.
- `sink.ImageJWriter` – for writing an `ImagePlusContainer` to a file format that ImageJ supports. If the image type cannot be determined based on the extension, you can also specify which type to generate.

Figure 2.1 shows a flow[2] for reading images, turning them into greyscale using a transformer and displaying them side-by-side. Figures 2.2 and 2.3 show original and greyscale image.

[1] `http://imagejdocu.tudor.lu/doku.php?id=faq:general:which_file_formats_are_supported_by_imagej`

[2] adams-imaging-transform_to_greyscale.flow

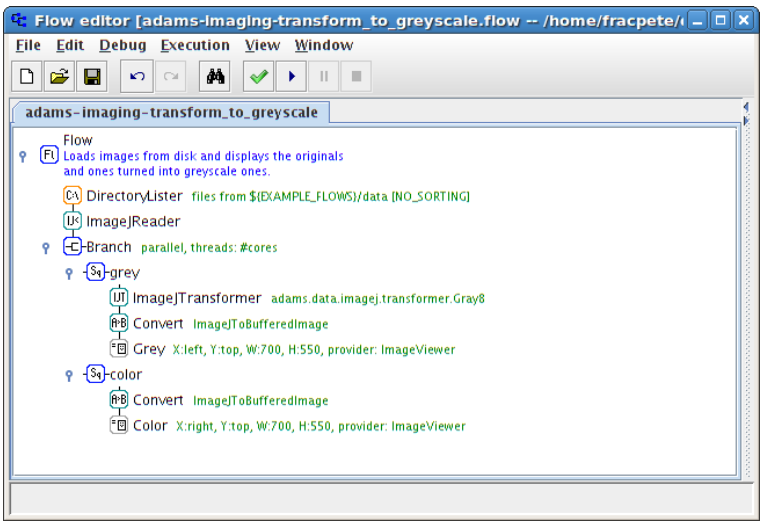

Figure 2.1: ImageJ flow for turning images stored in a directory into greyscale ones.

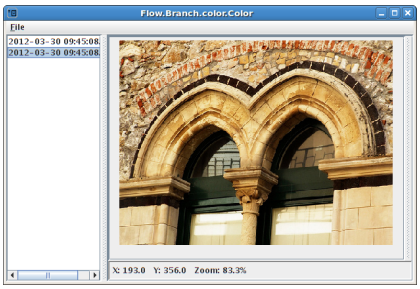

Figure 2.2: The original image.

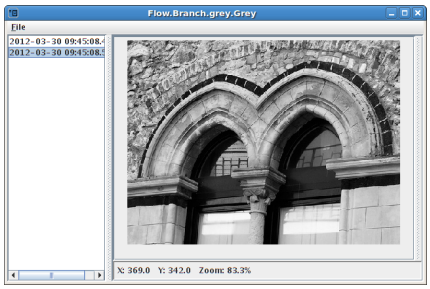

Figure 2.3: The greyscale image.

2.2 Plugins

By default, ADAMS includes plugins located in the following directory on Linux/Unix/Mac:

```
$HOME/.adams/imagej/plugins
```

and on Windows here:

```
%USERPROFILE%/_adams/imagej/plugins
```

You can override this directory by using the `ADAMS_IMAGEJ_DIR` environment variable, which defines the directory one level above the *plugins* directory. For instance, if your plugins directory is located at:

```
/home/user/imagej/plugins
```

You have to define the `ADAMS_IMAGEJ_DIR` environment variable as follows:

```
ADAMS_IMAGEJ_DIR=/home/user/imagej
```

Chapter 3

ImageMagick

ImageMagick® is a software suite to create, edit, compose, or convert bitmap images ([4]). On Windows, in order to process images with ImageMagick, you need to set the `IM4JAVA_TOOLPATH` environment variable, pointing to the installation.

There are three ImageMagick actors available:

- `transformer.ImageMagickReader` – for reading any image file that ImageMagick supports and forwarding a `BufferedImageContainer` object.
- `transformer.ImageMagickTransformer` – performs any ImageMagick command on the incoming image that the `convert` tool[1] supports and outputs another image again.
- `sink.ImageMagickWriter` – for writing a `BufferedImageContainer` to a file format that ImageMagick supports. If the image type cannot be determined based on the extension, you can also specify which type to generate.

There is no separate transformer for generating a WEKA instance, since the ImageMagick actors process and output `BufferedImageContainer` objects as well, just like the JAI actors. You can use the `JAIFlattener` for generating WEKA output.

The example flow[2] in Figure 3.1 loads a single photo from disk and then uses ImageMagick to resize it to 90 by 90 pixels and scaling it by 200% (see 3.2). Finally, the modified image is displayed in the image viewer.

[1] `http://www.imagemagick.org/script/convert.php`
[2] adams-imaging-imagemagick_script.flow

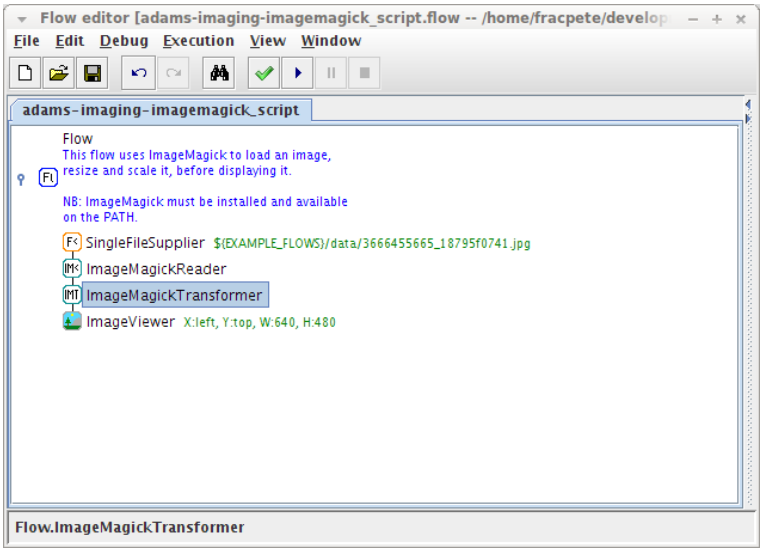

Figure 3.1: ImageMagick flow for processing (resizing) a single image.

```
# resizing image
-resize 90x90
# scaling it up again
-scale 200%
```

Figure 3.2: ImageMagick commands to resizing.

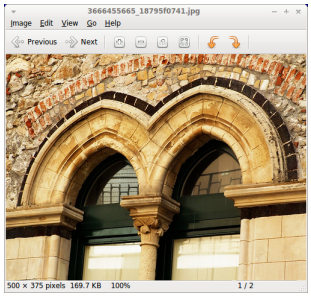

Figure 3.3: The original image.

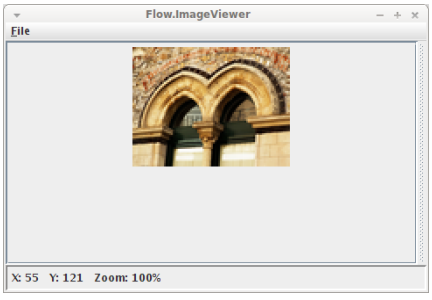

Figure 3.4: The resized image.

Chapter 4

Object conversion

JAI and ImageMagick actors generate and accept a different type of token, *BufferedImageContainer* namely, which cannot be processed by ImageJ actors. Vice versa, the tokens generated by ImageJ actors, of type *ImagePlusContainer*, are not accepted by JAI/ImagMagick actors. In order to exchange data between the two domains, the *Convert* transformer can once again be used.

The following conversions are available to convert from one format into another:

- *BufferedImageToImageJ* – for JAI/ImageMagick to ImageJ conversion.
- *ColorToHex* – turns a Color object into its hexa-decimal notation.
- *ImageJToBufferedImage* – converting from ImageJ to JAI/ImageMagick.
- *HexToColor* – turns a color in hexa-decimal notation back into a Color object.

Chapter 5

Interaction

The *PixelSelector* transformer allows the user to interact with the flow. The interaction with the user works as follows: an image viewer instance is displayed when the *PixelSelector* transformer receives an image token as input. The use then right-clicks on a pixel that he wants to process, e.g., labelling for WEKA data generation. After all the pixels have been selected and processed, the user then hits the *OK* button to close the dialog. The *PixelSelector* then forwards the image container with the attached, enriched report for further processing.

The *PixelSelector* transformer is very generic, which means the actor is responsible for the actions that the user can select from the right-click menu. This is done by selecting the appropriate actions from the list of available ones, e.g., *AddClassification* (package `adams.flow.transformer.pixelselector`), which is used for attaching classification labels to pixels. In order to make these selections visible not just in the report that is displayed on the right-hand side in the dialog, appropriate overlays can be selected as well, e.g., the *ClassificationOverlay* (package `adams.flow.transformer.pixelselector`) overlay, which displays the pixels with the associated labels on the screen.

Figure 5.1 shows a flow[1] that lets the user hand-label all JPG images in a directory and generated WEKA data from it. It uses a cropped region of 5x5 pixels around the selected pixels for the data generation. The user interface for selecting the pixels is shown in Figure 5.2 and a resulting dataset in Figure 5.3.

Of course, due to the interactive nature, labelling is performed on-the-fly and no record is kept. Once the image has been processed, the *PixelSelector* will forget about it. If you want to preserve the attached report, you can use the *ReportFileWriter* transformer to save the report to disk.

In order to re-use a previously saved report, you can use the *SetReportFromFile* transformer to replace the default report in the image container after you loaded the image with the one stored on disk. This allows you to continue work with previously generated labels, saving you a lot of work.

Since the *SetReportFromFile* transformer generates *ReportHandler* tokens, you need to explicitly cast the type of the tokens to the desired one, e.g., *BufferedImageContainer*, using the *Cast* control actor.

[1] adams-imaging-pixelselector.flow

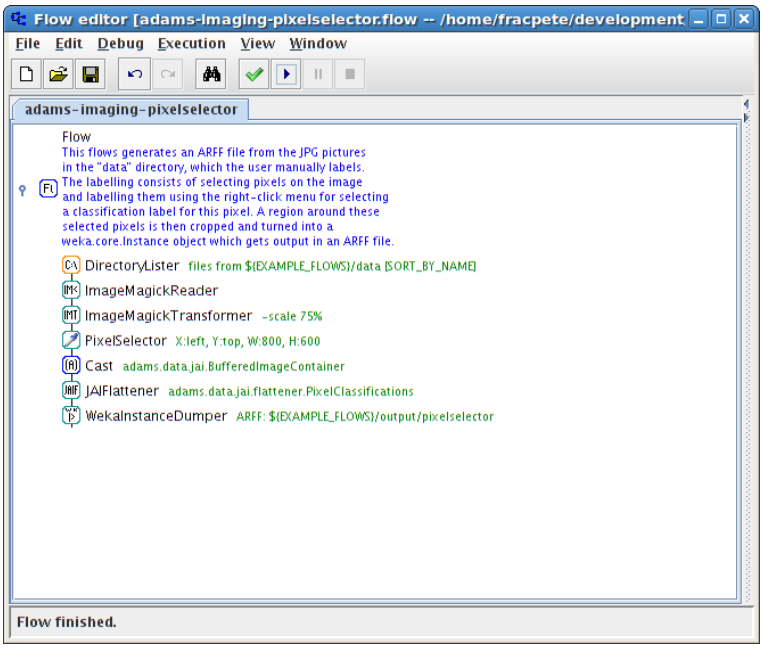

Figure 5.1: Flow for generating ARFF file from user-labelled pixels.

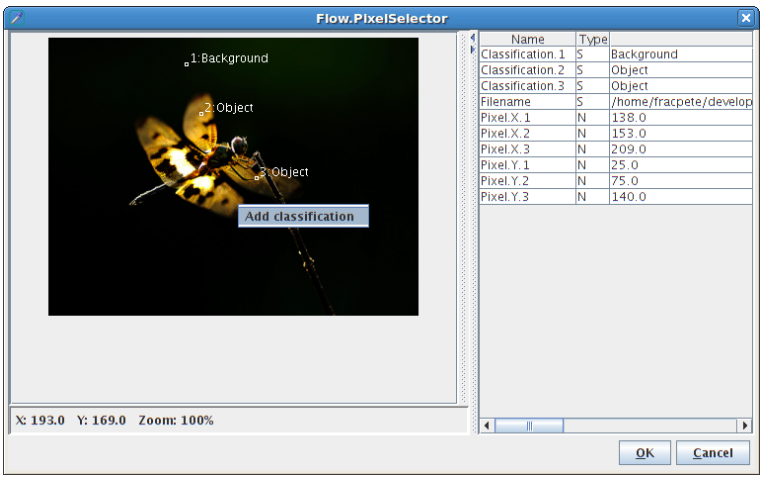

Figure 5.2: User interface for labelling pixels, displaying some pixels labelled already.

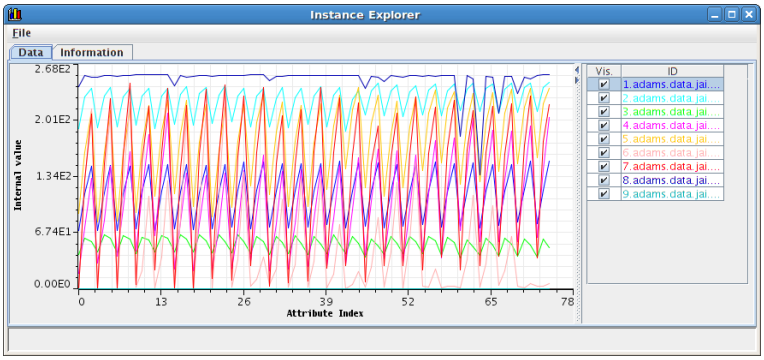

Figure 5.3: Example dataset generated using the PixelSelector.

Chapter 6

WEKA output

Of course, the data can be turned into a format that is suitable for WEKA ([6]). For JAI and ImageMagick transformers, both generating *BufferedImageContainer* tokens, the *JAIFlattener* can be used to generate WEKA output in the form of *weka.core.Instance* objects. For ImageJ generated tokens, outputting *ImagePlusContainer* tokens, you have to use the *ImageJFlattener* instead. This transformer also outputs *weka.core.Instance* objects. These *Instance* objects can then be processed further or simply dumped into a file. Figure 6.1 shows a flow[1] that generates an ARFF file from images using ImageJ. The resulting dataset, as displayed in the Instance Explorer, is shown in Figure 6.2.

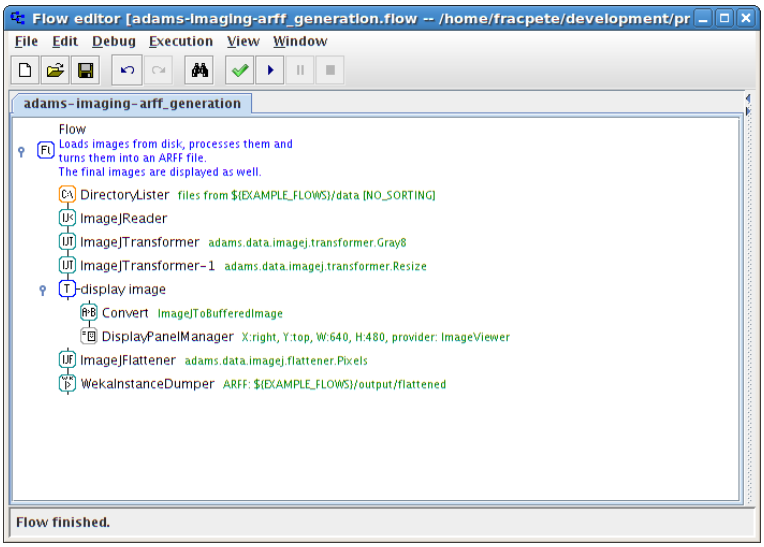

Figure 6.1: Generating an ARFF file using ImageJ.

[1] adams-imaging-arff_generation.flow

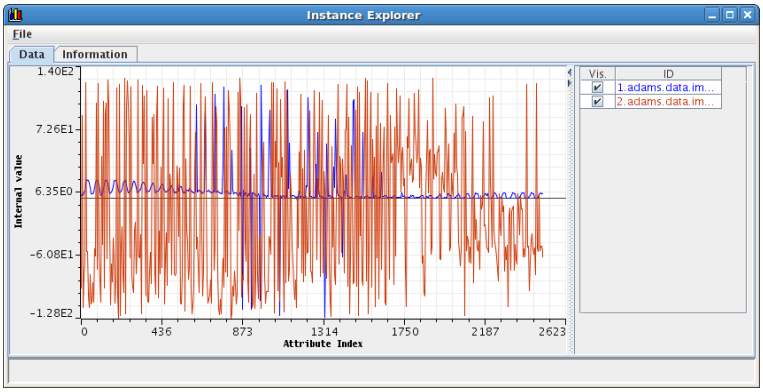

Figure 6.2: The ImageJ generated ARFF file.

Chapter 7

Miscellaneous actors

The imaging module offers some more actors that have not been introduced yet. Available sources:

- *ColorProvider* – outputs Color objects generated by a configured color provider.

Available transformers:

- *Draw* – Performs draw operations on images, like setting pixels, drawing lines, rectangles, ovals, text, images[1].
- *ImageInfo* – Allows you to obtain *width* and *height* information from an image.
- *ImageMetaData* – Extracts meta-data (EXIF or IPTC) from an image as spreadsheet.[2]

Available sinks:

- *FFmpeg* – actor for processing videos using ffmpeg[5][3].

[1] adams-imaging-draw.flow
[2] adams-imaging-meta_data.flow
[3] adams-imaging-ffmpeg.flow

Bibliography

[1] *ADAMS* – Advanced Data mining and Machine learning System
`https://adams.cms.waikato.ac.nz/`

[2] *JAI* – Java Advanced Imaging API
`http://java.sun.com/javase/technologies/desktop/media/jai/`

[3] *ImageJ* – Image Processing and Analysis in Java
`http://rsbweb.nih.gov/ij/`

[4] *ImageMagick* – Software suite to Convert, Edit, and Compose Images
`http://www.imagemagick.org/`

[5] *FFmpeg* – a complete, cross-platform solution to record, convert and stream audio and video
`http://ffmpeg.org/`

[6] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); *The WEKA Data Mining Software: An Update*; SIGKDD Explorations, Volume 11, Issue 1.
`http://www.cs.waikato.ac.nz/ml/weka/`