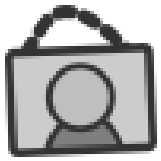
ADAMS

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

Module: adams-imaging



Peter Reutemann

November 21, 2026

©2009-2024 THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato


Except where otherwise noted, this work is licensed under
http://creativecommons.org/licenses/by-sa/4.0/

Contents

1 ADAMS **7**

2 Java Advanced Imaging **9**

3 LIRE **11**

4 Object conversion **13**

5 OCR **15**

6 Interaction **17**

7 Feature output **21**

8 Miscellaneous flow components **23**

Bibliography **27**

List of Figures

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

5.1 Preferences for tesseract. 15

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

Chapter 1

ADAMS

ADAMS has custom image processing support that does not rely on other libraries.

The following actors are available:

- `sink.ImageFileWriter` – writes an image container to a file using the specified writer.
- `transformer.BufferedImageTransformer` – performs a transformation using an existing transformer class on the incoming image and outputs another image again.
- `transformer.BufferedImageFeatureGenerator` – 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.
- `transformer.ImageFileReader` – reads an image file using the specified image reader.

Figure 1.1 shows a flow[1] 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] adams-imaging-gaussian_blur.flow

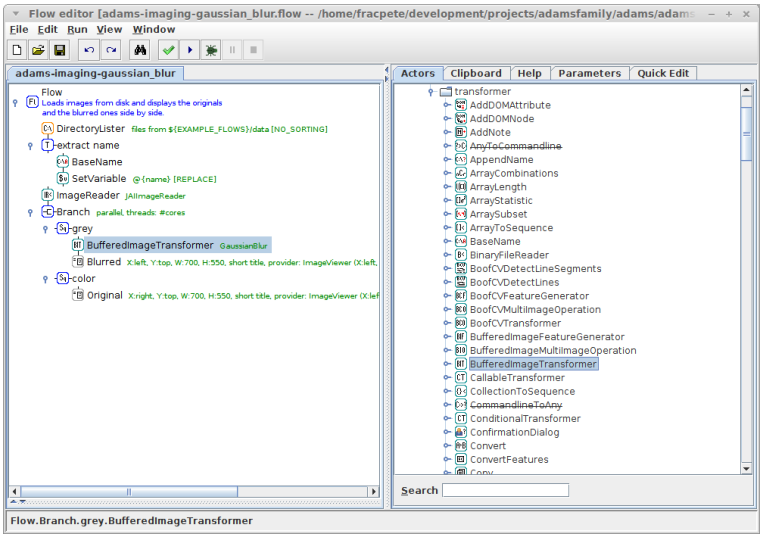


Figure 1.1: Flow for blurring images stored in a directory.

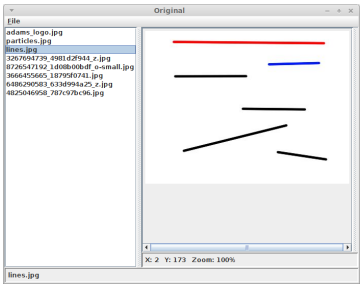


Figure 1.2: The original image.



Figure 1.3: The blurred image.

Chapter 2



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].

Reading and writing images are done using the *ImageFileReader* transformer and *ImageFileWriter* sink:

- `ImageFileReader` – use the *JAIImageFileReader*
- `ImageFileWriter` – use the *JAIImageFileWriter*

Since the JAI actors, readers and writers use `BufferedImageContainer`, the `BufferedImageTransformer` and `BufferedImageFeatureGenerator` transformers can be used.

[1] `http://en.wikipedia.org/wiki/Java_Advanced_Imaging`

Chapter 3

LIRE

The Lucene Image Retrieval library [4] provides a wide range of feature generators that work on *BufferedImageContainer* objects:

- AutoColorCorrelogram
- BasicFeatures
- BinaryPatternsPyramid
- CEDD
- ColorLayout
- EdgeHistogram
- FCTH
- FuzzyColorHistogram
- FuzzyOpponentHistogram
- Gabor
- JCD
- JpegCoefficientHistogram
- LocalBinaryPatterns
- LuminanceLayout
- OpponentHistogram
- PHOG
- RotationInvariantLocalBinaryPatterns
- ScalableColor
- SimpleColorHistogram
- Tamura

Chapter 4

Object conversion

The following conversions are available:

- *ColorToHex* – turns a Color into its hexa-decimal notation.
- *HexToColor* – turns a color in hexa-decimal notation back into a Color object.
- *LocatedObjectToRectangle* – extracts the location of the object and stores it in a rectangle object.
- *LocatedObjectsToReport* – converts the array of LocatedObject instances into a Report.
- *ObjectAnnotationsToImageSegmentationLayers* – converts the object annotations into layers for image segmentation.
- *QuadrilateralLocationCenter* – outputs the center of the QuadrilateralLocation object.
- *QuadrilateralLocationToString* – turns the QuadrilateralLocation object into its string representation.
- *RectangleCenter* – outputs the center of the rectangle object.
- *RectangleToString* – converts the rectangle object into its string representation.
- *StringToQuadrilateralLocation* – creates a QuadrilateralLocation object from this string representation.
- *StringToRectangle* – creates a rectangle object from the string representation.

Chapter 5

OCR

A common task in image processing is *optical character recognition* (OCR). ADAMS offers a simple wrapper around the open-source *tesseract* engine [7]. The engine is available for Windows, Linux and Mac OSX. It supports multiple languages, however, these need to be installed in order to be actually available.

The follwoing actors are available:

- *TesseractConfiguration* – standalone for configuring OCR, mainly to define where the tesseract executable is located.
- *TesseractOCR* – this transformer turns an image file into one or more text files, which need to be further processed in the flow then.[1]

By default, the *TesseractConfiguration* standalone uses the globally defined preferences as default values. In the preferences dialog (*Main menu* → *Program* → *Preferences* → *Tesseract*) you can specify the location of the tesseract executable and the default language (see Figure 5.1).

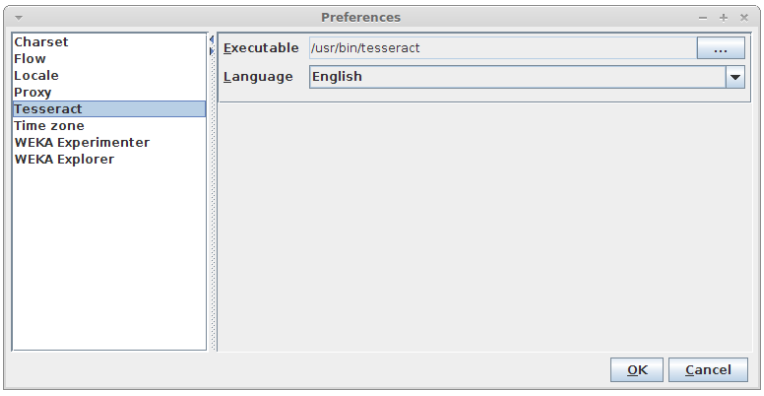


Figure 5.1: Preferences for tesseract.

[1] adams-imaging-ocr.flow

Chapter 6

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 6.1 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 6.2 and a resulting dataset in Figure 6.3.

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* or *SetReportFromSource* 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* and *SetReportFromSource* transformers generate *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

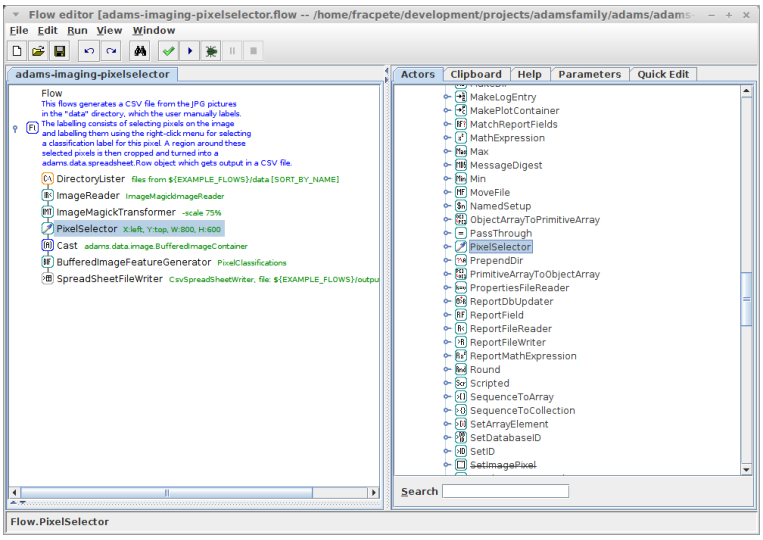


Figure 6.1: Flow for generating ARFF file from user-labelled pixels.

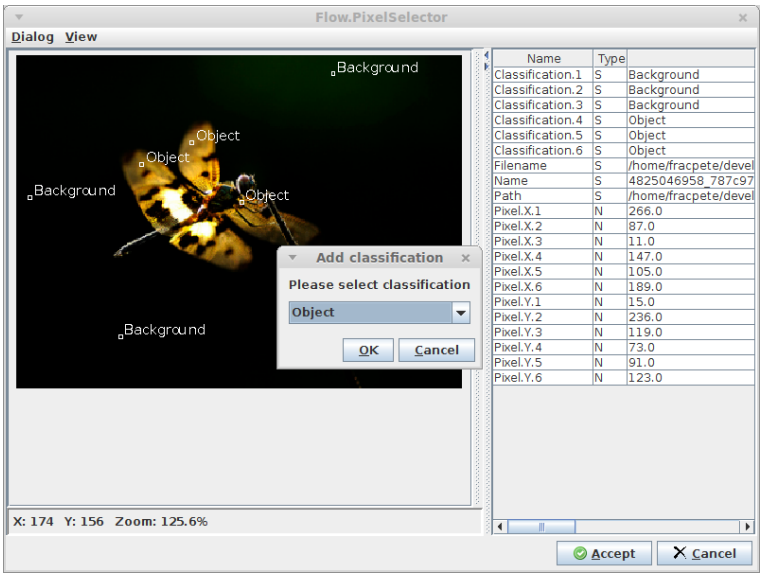


Figure 6.2: User interface for labelling pixels, displaying some pixels labelled already.

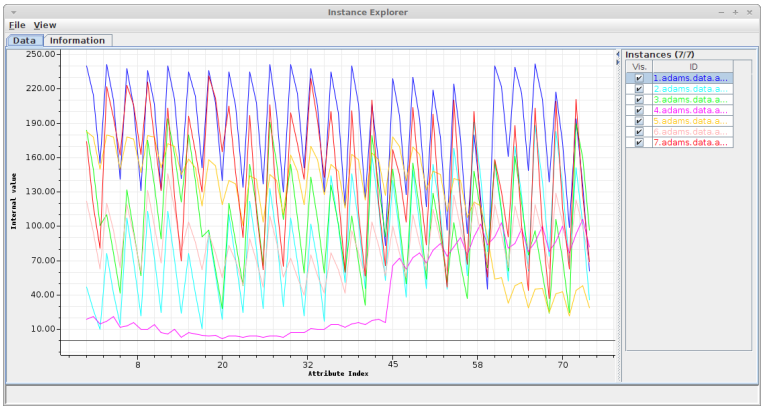


Figure 6.3: Example dataset generated using the PixelSelector.

Chapter 7

Feature output

Of course, the data can be turned into a format that is suitable for machine learning applications like WEKA ([9]). For JAI transformer tokens, the *BufferedImageFeatureGenerator* can be used to generate such output.

Chapter 8

Miscellaneous flow components

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

- *HasExifTag* – checks whether the specified tag is present in the data coming through.

Available sources:

- *ColorProvider* – outputs Color objects generated by a configured color provider.
- `NewImage` – creates an image with a specific color and user-defined dimensions.

Available transformers:

- *ChangeImageObjectPrefix* – replaces the prefix of objects in a report with a new one.
- *ColorProvider* – outputs Color objects whenever a token passes through.
- `CompareObjectLocations` – for visually comparing annotations and predicted object locations for an image (side-by-side or combined).
- *CountObjectsInPolygon* – counts the objects that fall within the defined polygon (partial counts possible, too).
- *CountObjectsInRegion* – counts the objects that fall within the defined rectangular region (partial counts possible, too).
- *DecodeBarcode* – allows extracting of barcodes like EAN and QRCode from images[1].
- `DetermineOverlappingAreas` – computes the areas of overlapping image objects between two reports (annotations and predictions).
- `DetermineOverlappingObjects` – computes overlapping image objects between two reports (annotations and predictions).

[1] adams-imaging-barcode.flow

- *Draw* – Performs draw operations on images, like setting pixels, drawing lines, rectangles, ovals, text, images[2] and even barcodes (e.g., EAN and QRCode)[3].
- *EncodeBarcode* – allows turnings strings into barcodes like EAN and QR-Code from images[4].
- *ExifTagOperation* – allows operations on EXIF meta-data tags.
- `GetImageObjectIndices` – lists all the indices of the objects stored in the report, as located by the specified finder.
- `GetImageObjects` – lists all the objects stored in the report, as located by the specified finder.
- `GetImageObjectMetaData` – retrieves the meta-data of the located object.
- *ImageInfo* – Allows you to obtain *width* and *height* information from an image.
- `ImageLabeler` – allows the user to interactively label/classify images, attaching the label to the report. annotate objects (i.e., attach labels) that were located in an image.
- *ImageMetaData* – Extracts meta-data (EXIF or IPTC) from an image as spreadsheet using various libraries (e.g., Sanselan[6], Meta-Data Extractor[5]).[5]
- `ImageObjectAnnotator` – allows the user to interactively annotate objects (i.e., attach labels) that were located in an image.
- *ImageObjectFilter* – filters the objects in the meta-data of an image using the specified object finder and filter.
- `ImageObjectInfo` – extracts information from a image object stored in a report.
- `ImageObjectIndexOffset` – changes the index of the objects in the report by the specified offset.
- `ImageObjectOverlap` – superceded by *DetermineOverlappingObjects*.
- `ImageSegmentationAnnotator` – allows the user to interactively annotate images for image segmentation.
- `ImageObjectOverlapMatrix` – computes a matrix of overlapping objects between annotations and predictions, which can be turned into a confusion matrix.
- `ImageObjectToVariables` – turns the properties of the image object (of type `LocatedObject`) passing through into variables.
- `ImageSegmentationContainerFilter` – allows the user to apply a filter to an image segmentation container.
- `ImageSegmentationContainerOperation` – allows the user to apply operations to image segmentation containers.
- `ImageSegmentationFileReader` – for reading image segmentation file formats
- *ImageSharpness* – determines whether image is in focus or not.
- `IntersectOverUnion` – superseded by *DetermineOverlappingObjects*.
- *LocateObjects* – provides a framework for algorithms that locate objects in images.

[2] adams-imaging-draw.flow
[3] adams-imaging-barcode.flow
[4] adams-imaging-barcode2.flow
[5] adams-imaging-meta_data.flow

- `MergeObjectLocations` – merges objects locations in the current image container with the ones obtained from a report available from storage.
- `NegativeRegions` – generates negative region objects for images passing through.
- `RemoveImageObject` – removes the specified image object from the report.
- *RemoveOverlappingImageObjects* – for cleaning up overlapping objects, e.g., as post-processing from predicting bounding boxes.
- `ScaleReportObjects` – scales the x/y and width/height of the objects stored in reports.
- `SetImageObjectMetaData` – sets/updates the meta-data of a located object.
- `ViaAnnotations` – turns reports into JSON annotations that can be used by VIA[8].
- `ViaAnnotationsToReports` – turns VIA JSON annotations into separate reports, one per annotated image.

Available sinks:

- *ImageHistogram* – displays the histogram of an image (gray or color).
- `ImageSegmentationFileWriter` – for reading image segmentation file formats
- `ImageSegmentationViewer` – for viewing image segmentation containers
- `ObjectLocationComparison` – for visually comparing annotations and predicted object locations for an image (side-by-side or combined).

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

[4] *LIRE* – Lucene Image Retrieval
`http://code.google.com/p/lire/`

[5] *MetaData-Exrtactor* – Library for reading metadata from image files.
`https://code.google.com/p/metadata-extractor/`

[6] *Sanselan* – Apache Imaging (formerly known as Sanselan)
`http://commons.apache.org/proper/commons-imaging/`

[7] *tesseract* – An OCR Engine that was developed at HP Labs between 1985 and 1995... and now at Google.
`http://code.google.com/p/tesseract-ocr/`

[8] *VIA* – VGG Image Annotator
`http://www.robots.ox.ac.uk/~vgg/software/via/`

[9] 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/`