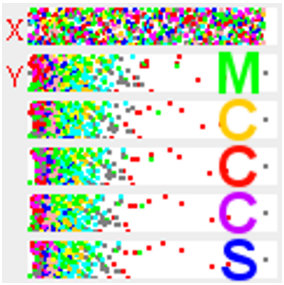
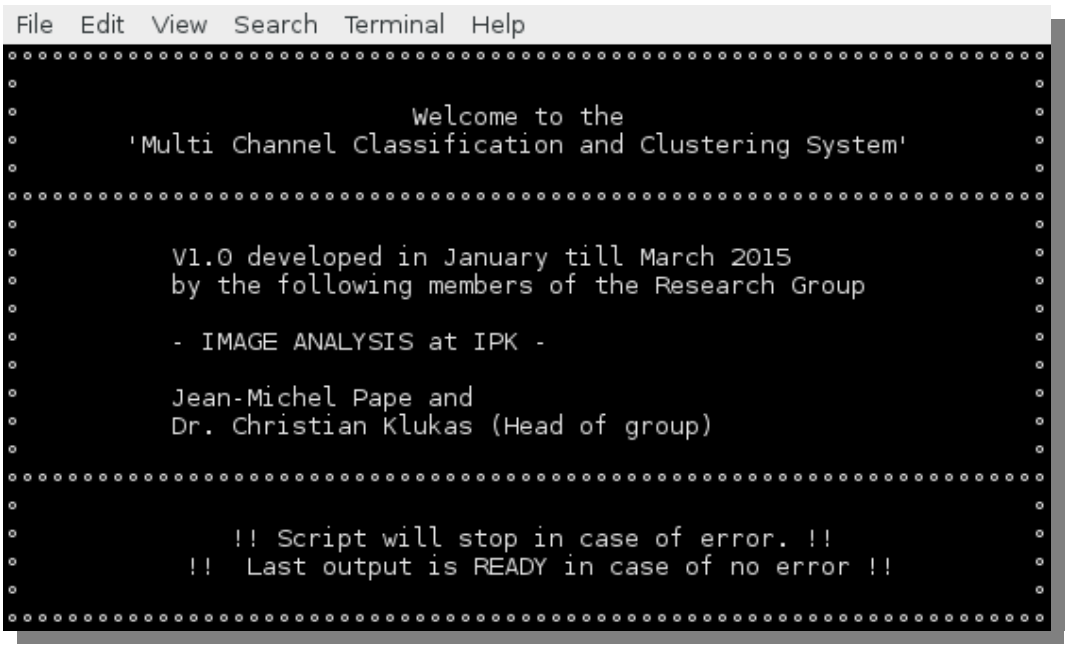
MCCCS

Multi Channel Classification and Clustering System

User Documentation for MCCCS

Jean-Michel Pape & Christian Klukas





Last update: 21.01.2016

Table of contents

Overview

The MCCCS (Multi Channel Classification and Clustering System) has been developed by the group Image Analysis (Jean-Michel Pape & Christian Klukas) at IPK (Leibniz Institute of Plant Genetics and Crop Plant Research, D–06466 Gatersleben, Germany). It is a powerful system which utilizes machine learning approaches for image processing and image analysis. Especially, it is designed to solve segmentation tasks. Furthermore, it is able to handle multi channel data, such as hyperspectral data sets (BSQ, TIFF) and supports up to 32-bit images. The example applications illustrates the main capabilities and the usage of the system. The examples can be easily downloaded and prepared (see Chapter 'Run application examples').

Installation

General notes

MCCCS can be downloaded from the following sourceforge project website: http://sourceforge.net/projects/mcccs/. Additional information is also provided on the project page http://mcccs.sourceforge.net/. The provided .zip file includes the utility **commands** for image processing and file conversion, which are included into the *mcccs.jar*. Preferred machine learning libraries can be individually selected by the user (here, for the application examples, the **WEKA** library is used). Also **example scripts and datasets** are included to show the systems main capabilities, within different application examples. Usually, several commands are summarized in a script file which represents a so called **pipeline**.

All commands, included in the mcccs.jar, can be run by using a command-line interface (terminal). Commands are called using the following scheme.

$JAVA.command parm_1 parm_n file

A detailed overview about the used parameters for each command can be found in the appendix.

Please see next section for detailed system requirements.

System requirements

For using the system a commandline interface like GNU-Bash (linux, MAC) or Cygwin (Windows) and a Java installation (Java Runtime Environment, JRE of version 1.8 or higher, sometimes called Java 8 or higher) is needed[1]. Preferred are 64-bit versions of Java, as they support the utilization of more than 1600 MB of RAM. Machine learning libraries can be selected by the user, we recommend WEKA (http://www.cs.waikato.ac.nz/ml/weka/). Also a commandline interface like Bash is needed.

LINUX → Tested on Fedora Core 20 and Arch Linux using Bash.

MS WIN → All scripts will be made available using the Cygwin commandline interface (https://www.cygwin.com/).

MAC → Tested on Mac OS 10.10 (Yosemite) and 10.11 (El Capitan)

1 MAC users, please install the Java Development Kit JDK instead of the JRE.

Run application examples

Preparation

Installation of additional commands

For running the provided application examples, a prepare script is provided (downloading the needed data sets and libraries). Therefore, the Wget[2] command is needed (http://www.gnu.org/software/wget/) for data transfer purposes. Also, UnZip[3] (http://www.info-zip.org/) is utilized for archive extraction. Also as mentioned in the system requirements a commandline interface and Java installation is needed. Depending on the system configuration, the installation of additional packages or other versions may be needed. To check the command availability and version on your specific system, please start the corresponding commands in the commandline interface (WIN → cygwin; LINUX, MAC → terminal). May add the additional option --version.

Example for the bash command:

bash --version

output:

GNU bash, version 4.2.53(1)-release (x86_64-redhat-linux-gnu) ...

It is also recommended to check the following commands:
→ ***wget***, ***bash***, ***unzip***, ***bc***, ***xargs*** and ***java***

If no error appears, your system contains the required commands. Most of the Linux distributions includes the needed shell commands. Mac and Windows/Cygwin users, please note the following hints to install the needed commands. Cygwin users also need the **unzip**, **wget** and **bc**[4] command, which is not included in the default installation.

Additional hints for Mac users

- for running **Java** in the terminal, the Java Development Kit is needed
- the **Wget** command can be installed by using Rudix (http://rudix.org/packages/wget.html).

2 Tested with version 1.16.1.
3 Tested with version 6.0.
4 bc is an arbitrary precision calculator language

Additional hints for Windows/Cygwin users

- for Cygwin it is recommend to choose the needed packages (**UnZip**, **Wget**, **bc**) by utilizing the package search tool while installation (in an existing installation just restart the setup to add additional packages)

Download and preparation of the example data

The application examples can be downloaded and prepared by executing the *prepare_datasets.sh* in a terminal. The example data and needed libraries are automatically downloaded and transferred into the common folder structure for processing with the given example scripts. Included are the following examples:

Application example descriptions

1. segmentation_example_1_classification

This example shows an application for foreground/background segmentation for top view plant images (Arabidopsis thaliana - A1, A2 and tobacco - A3) using a supervised Random Forest classifier. The three data sets are split into a training set and a data set for prediction as well. After processing, the segmentation results, named *foreground.png,* are stored in each sub folder, e.g. *plant_003*.

2. hyper_example_1_classification

This example shows an application for a multi-labeled segmentation on an airbone hyper-spectral image data set. Here partly pre-classified ground-truth image masks are used to train a supervised Random Forest classifier. After processing, the segmentation result, named *classified.png* is stored in the experiment sub folder (*stack_images → dc*).

3. hyper_example_2_clustering

This example shows an application for a multi-labeled segmentation on an airbone hyper-spectral image data set as used in the example before. Instead of using pre-classified ground-truth data to train a supervised classifier here a clustering approach is performed. After processing, the segmentation result, named *clustered.png* is stored in the experiment sub folder (*stack_images → dc*).

Start of the analysis ...

The analysis can be started by navigating into the corresponding experiment folder, by executing the *process_ … .sh* script in a terminal (e.g. segmentation_example_1_classification → execute *process_segmentation_example_1_classification.sh* in the experiment folder). The results, including a labeled result image and the belonging numeric data, named *all_ … .csv,* are stored into the corresponding sub-folders.

Detailed analysis workflow description: Segmentation example using supervised classifier

Here a example workflow description is presented. It describes the general procedure to perform a foreground/background segmentation using a supervised classification approach. The workflow includes the following steps:

1. Creation of training data

2. Data preparation (create folder structure)

3. Script adaptation for training and prediction

This guide provides an exemplary overview about the general procedure and the main components (commands) for solving a common segmentation task. For a detailed understanding it is recommended to become familiar with the provided application examples by studying the given example pipeline-scripts.

Input data

For the illustration of the example workflow a top-view image including an '*arabidopsis thaliana*' plant is used. The goal of the analysis is to generate a foreground/background segmentation utilizing a supervised classification approach[5].

5 example data and used plant images within the documentation are from: http://www.plant-phenotyping.org/CVPPP2014-challenge, please cite this paper if you use this data for your research:

Hanno Scharr, Massimo Minervini, Andreas Fischbach, Sotirios A. Tsaftaris. Annotated Image Datasets of Rosette Plants. Technical Report No. FZJ-2014-03837, Forschungszentrum Jülich, 2014).

Illustration 1: RGB input image of an arabidopsis plant. (see http://www.plant-phenotyping.org/CVPPP2014-challenge)

Data preparation

Training-data preparation

A supervised classification use training data to create a model which includes some knowledge to predict the un-classified data. First samples for foreground and background have to be created utilizing an image editing software e.g. GIMP (http://www.gimp.org/). The following illustrations show the common proceed, it is sufficient to highlight only a few spots of the foreground and background regions (see illustration below). The labeled images could be named as *mask_1 … n.png*.

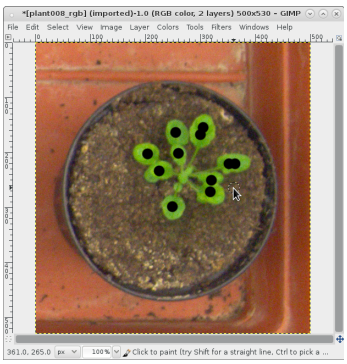


Illustration 2: Labeling of the foreground using GIMP.

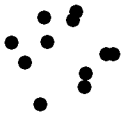


Illustration 3: Marked foreground regions.

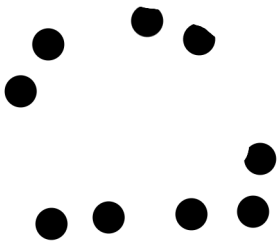


Illustration 4: Manually marked background mask.

Creating folder structure

The underlying folder structure is not strict designed, it depends on the utilized scripts. Here the structure used in the application examples is recommended. This exemplary structure looks like this:

```
→ experiment
      → script files (e.g. preprocess.sh, train.sh, predict.sh, ...)
      → training
            → replicate_1
                  → image
                  → mask_1 … n
            → replicate_2
            → replicate_n
      → prediction
            → replicate_1
                  → image_1 … n
            → replicate_2
            → replicate_n
```

For generation of the sub folder structure some scripts are provided, e.g. *move_all_to_subdir.sh* can be used to move all replicate images (which may be stored together into one folder) into its belonging sub-folder.

Training

Now the analysis scripts have to be prepared. The training script includes following steps:

Split RGB

SplitRGB *image*

→ Splits an RGB input image into separate channel images.

ARFF sampling

ArffSampleFileGenerator number_of_channels number_of_classes

sample_size folder

(e.g. for segmentation use following parameters: 3, -2, 2000, replicate_i)

→ Sampling and feature selection from input images, generates .arff file for classifier training.

The individual .arff files could be summarized using commadline utilities, e.g.:

```
for dir in */;
do
     cat "${dir}/fgbgTraining.arff" | grep -v filter | grep -v
     another_filter   >> all_fgbg.arff
done
```

Here all *fgbgTraining.arff* files from the sub-directories are summarized to an file named *all_fgbg.arff*. This file will be used for classifier training.

Model training

The following command depends on the used machine learning library, for our example using WEKA we could define a Random Forest classifier as follows:

$WEKA weka.classifiers.meta.FilteredClassifier -t 'all_fgbg.arff' -d fgbg.model -W weka.classifiers.trees.RandomForest -- -I 100 -K 0 -S 1

Here the summarized ARFF files *all_fgbg.arff* are loaded by the WEKA library, the result is a model named fgbg.model which can be used to predict the fore- and background on the un-classified images.

In case of fully labeled ground truth data some scripts for evaluation are provided such as:

CreateDiffImage ground_truth classification_result

The script generates a difference image between the prediction and the ground truth mask.

Prediction

Image conversion

For the prediction of the whole un-classified image, first it has to be converted into its belonging ARFF file using the following command:

ArffFromImageFileGenerator number_of_channels number_of_classes folder

Apply trained model

Utilizing WEKA the trained model 'fgbg.model' can be applied on a unclassified ARFF file using:

$WEKA weka.filters.supervised.attribute.AddClassification -i "${dir}/${dir}.arff" -serialized fgbg.model -classification -remove-old-class -o "${dir}/result.arff" -c last

Using the previously generated model and the converted image ARFF file, a result ARFF file including the classification information will be generated.

Classification result mask creation

The generated result ARFF file is used to create an result mask including the segmentation result. This mask defines all pixels which belongs to the fore- and background. By applying this mask on the input image the final segmentation can be created.

ApplyClass0ToImage output_name

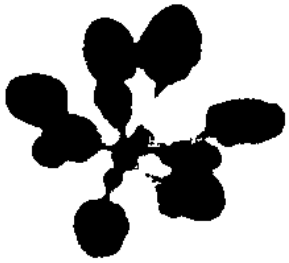


Illustration 5: Generated result mask including the labeled foreground (black).

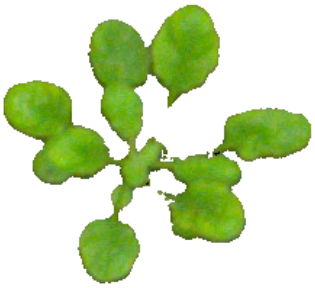


Illustration 6: Result of applying mask on input image.

Post processing

For quantification of the result image the quantify command can be used. It creates automatically a CSV file for each individual processed data file.

Quantify result_image

This will create a output CSV file like this:

File	Class_1	Class_2	overall
Repl_1	x	y	z

Here class_1 and class_2 represent the fore- and background. To summarize all CSV files a script, as shown in the following box, could be utilized which summarize all individual CVS files. Afterwards the the CVS file is transformed by the *TransformCVS* command.

```
for dir in plant*/;
do
     cat ${dir}/*_quantified.csv >> all_results.csv
done
$JAVA.TransformCSV all_results.csv
mv all_results.csv.transformed all_results.csv
```

APPENDIX

Command description

ArffFromImageFileGenerator

Prameter(s): [channel-count], [class-count], [filenames]

Creates a ARFF file from a list of images, representing the different input channels.

ArffSampleFileGenerator

Prameter(s): [channel-count], [[-]class-count, negative for FGBG separation], [sample-size], [filenames]

Sampling and feature selection from input images, generates .arff file for classifier training.

ArffToImageFileGenerator

Prameter(s): [channel-count], [filenames]

Recreates an image file from a labeled ARFF file. If in addition a mask image is provided, only foreground pixels from the mask are processed and filled with pixels, colored according to the ARFF file sample class information.

ApplyClass0ToImage

Prameter(s): [filename]

Generates a labeled result image in case of foreground/background segmentation.

ExportImagesFromHyperspec

Prameter(s): [prefix], [overflow threshold or negative value to disable], [filenames]

Exports each channel of an hyperspectral data set in a separate image. Ready for processing the BSQ format, also tif stacks including the channel-data in different slices.

SplitTiffStackToImages

Prameter(s): [filename]

Splits Tiff stack into individual images (for hyper-spectral data).

SplitRGB

Prameter(s): [filename]

Splits a given RGB input image into a set of three gray-scale images (R/G/B), representing the different input channels.

PowerSetGenerator

Prameter(s): [class-count], [filenames]

Processes a list of given input images, containing labeling information on a particular disease class. The combinations (multi-labeling problem) of overlapping masks are calculated and according result label images are generated.

Quantify

Prameter(s): [filename]

Calculates the areas of the different labels in the input image and creates a corresponding CSV result file.

RGB2HSV/LAB/XYZ

Prameter(s): [filename – R], [filename – G], [filename - B]

Converts the given input RGB data into spcified color-space.

Filter

Prameter(s): [filenames], [output-filename], [masksize], [sigma], [operation mode]

Enables different kinds of image filters (operation modes) (Gaussian Blur (sigma), Median, Sharpen, Texture) for enhanced feature calculation. The parameter masksize defines the filter-size.

SmoothSides

Prameter(s): [filenames]

Fits polynoms to the foreground object left and right sides, and creates a result image with a smoothed representation of the input image.

SplitObjects

Prameter(s): [filenames]

Objects in the input image, which reach from top to bottom and which are separated horizontally from each other are split from each other. A list of output images is created, each containing a single object extracted from the input image.

CreateDiffImage

Prameter(s): [filenames]

Generates difference image between calculated result and given ground truth image for evaluation purposes.

TransformCSV

Prameter(s): [filenames]

Combines multiple CSV input image files, with information on different labels into a single table, with a overall set of label-columns.