# Documentation of D4.4: SILKNOW Image Classification

*Release 0.0.1*

**LUH**

**Dec 17, 2019**

# CONTENTS

CHAPTER

ONE

# OVERVIEW OF THE DOCUMENTATION

This software provides python functions for the classification of images and training and evaluation of classification models. It consists of five main parts: The creation of a dataset, the training of a new classifier, the evaluation of an existing classifier, the classification of images using an existing classifier and the combined training and evaluation in a five-fold cross validation. All functions take configuration files as an input and generally write their results in specified paths. The format required for the configuration files is described in the SILKNOW Deliverable D4.4.

**The requirements for the silknow_image_classification toolbox are python 3.7.4 and the following python packages:**

- numpy
- urllib3
- pandas
- tqdm
- tensorflow (1.13.1)
- tensorflow-hub (0.6.0)
- matplotlib
- sklearn
- xlrd

# CHAPTER TWO

# SILKNOW IMAGE CLASSIFICATION

## 2.1 Summary of the silknow_image_classification Toolbox

All functions in the image classification package as well as a short description of them are listed in the following table. Afterwards, the operating principle of each function is explained and the respective input and output parameters are described.

| Name of the function | Short description of the function |
|---|---|
| CreateDataset | Creates a dataset with samples for the training and evaluation (Data Preparation). |
| train_CNN_Classifier | Trains a new classification model. (Training) |
| evaluate_CNN_Classifier | Evaluates an existing classification model. (Evaluation) |
| crossvalidate_CNN_Classifier | Trains and evaluates a new model using cross validation. |
| apply_CNN_Classifier | Classifies images using an existing classification model. (Classification) |

Table 1: Names and short descriptions for the functions of the SILKNOW image classification software.

Figure 1 shows the process flow of the image classification software and how it is embedded in the SILKNOW project.

### 2.1.1 CreateDataset

The goal of this function is to initialize the set of training samples from data exported from the knowledge graph and a CNN configuration file, both of which constitute important input datasets for all subsequent processes. For convenience, it also exports text files with the default control parameter values for the other four functions. An input file *DatasetConfiguration.txt* containing the control parameters must exist in the current working directory. The following parameters need to be defined in the *DatasetConfiguration.txt* file:

- **InputDataPath**: Path to the folder containing the files required for the dataset generation, i.e. the .CSV files containing the data extracted from the knowledge graph and the mapping table.
- **ImageCSV**: Name of the .CSV file containing the image URLs. This file must be stored in the path defined by InputDataPath.
- **FieldCSV**: Name of one of the .CSV files containing the raw samples with information about one of the variables. This file must be stored in the path defined by InputDataPath.
- **MappingTable**: Name of the .xlsx file that defines the class structure and the mapping from raw values to refined classes. This file must be stored in the path defined by InputDataPath.
- **OutputDataPath**: Path to the folder where the collection files and input images will be stored.
- **minNumSamples**: The minimum number of samples that should occur for a single class. Classes with fewer samples will not be considered by the prediction.
- **onlyFromCollection**: Restrict the dataset to only contain samples from one specific collection.

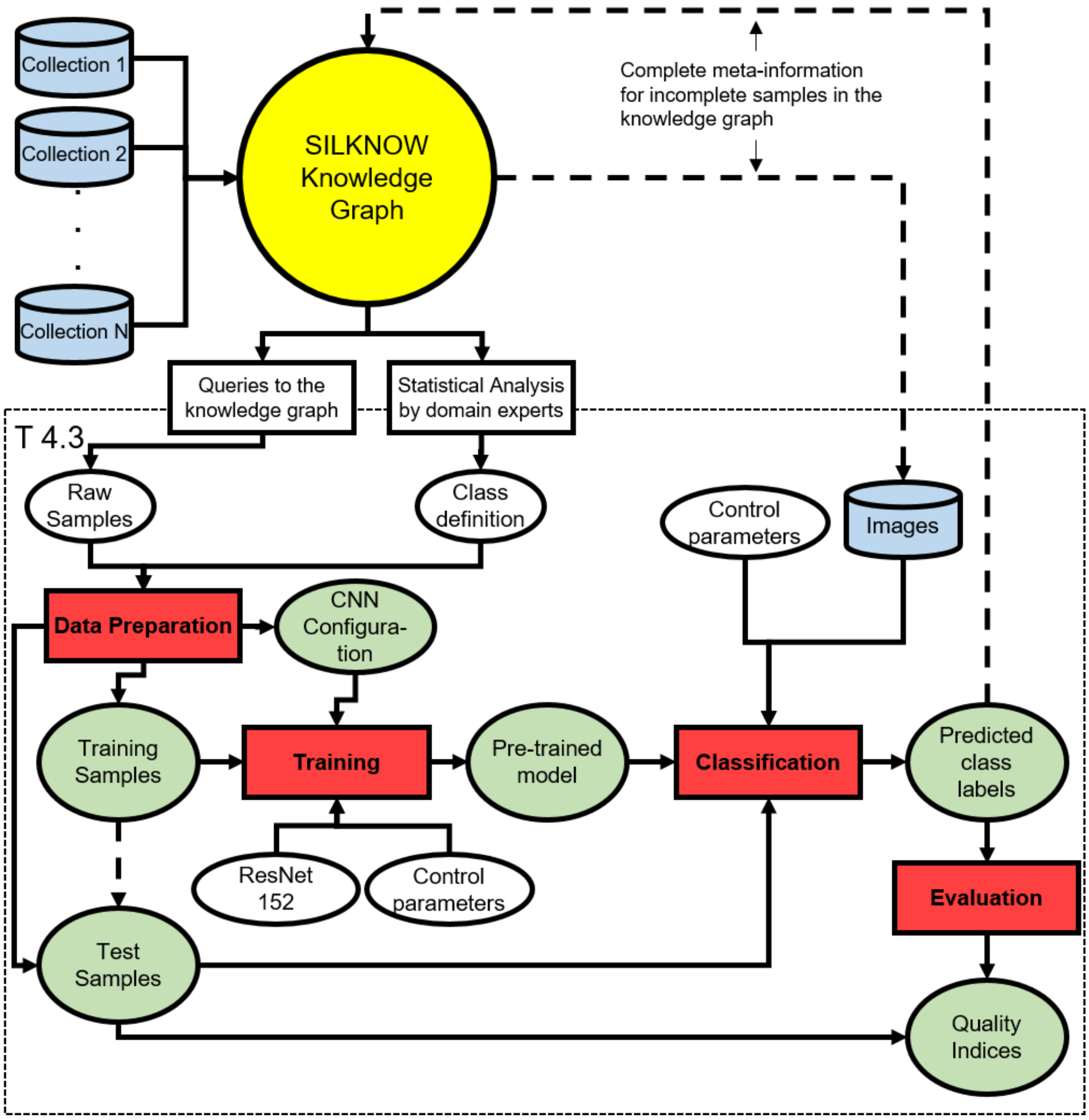

Fig. 1: Process flow of the SILKNOW image classification module and its embedding in the context of SILKNOW.

- **num_joint_fc_layer**: Number of shared fully connected layers in the network architecture.
- **num_nodes_joint_fc**: Number of nodes in each shared fully connected layer in the network architecture.

The input file has to be formatted in the following way. Each line of the text file defines one single parameter. A line starts with a parameter name as given in the list above, which is followed by a semicolon. The value for the parameter has to be placed behind the semicolon. For the parameter FieldCSV, there are two values, so that the row has the format: 'FieldCSV; file_name; field_name' It is important that *field_name* is the same as the field name in the header of the .CSV file *file_name* and that those field_name also is identical to the name of one of the sheets of the mapping table defining the class structure. There is one such row for every variable to be predicted. An exemplary input file containing control parameters can be found in the Git project at */silknow_image_classification/DatasetConfiguration.txt*. The output of the function 'CreateDataset' is three-fold:

- A **CNN configuration file**, i.e. a text file *CNNConfigurationFile.txt* that will contain the list of variables to be predicted and the corresponding class structure. This CNN configuration file is used to generate the CNN structure in the Training process. It starts with the hyperparameters describing the structure of the shared layers of the CNN. Then it contains one row per variable to be predicted. Each row starts with the keyword variable_and_class followed by a semicolon (";"). This is followed by the name of the variable as it is referred to in the knowledge graph. After that, each row contains a list of all possible class names. All variable and class names are preceded by the character "#". An exmaple for such a file can be found at */silknow_image_classification/CNNConfiguration.txt*.
- The **training samples**, which are distributed over several files. The training samples will be stored in a user-defined output directory 'OutputDataPath'. Firstly, the program CreateDataset generates a subfolder img of the output directory, and it downloads all images to the new directory 'OutputDataPath/img'. The filename of the image corresponding to the sample having the identifier ID will be ID.*ext*, where *ext* is the extension of the original image data file that indicates the file format. After that, the set of training samples is randomly split into five subsets of approximately equal size. This split facilitates training and evaluation by five-fold cross validation for scientific purposes and serves no other purposes. The metadata of the training samples will be stored to five so-called collection files (one per subset; the naming indicates that each contains a collection of training samples). Finally, there is a master file that contains the names of all collection files (one per row). Both the master and the collection files are generated in the directory 'OutputDataPath'. The collection files contain a header row describing the individual fields. Each field is identified by its name preceded by the character "#"; the individual field names are separated by tabs ("t"). The first field is the image file name referred to as *#image_file*, all other fields correspond to the variable names. After the header row, each row corresponds to one sample. It lists the image file name and the labels for all variables in the order indicated by the header row. Again, individual records are separated by tabs ("t"). An example for such a file can be found at */silknow_image_classification/samples/collection_1.txt*.
- **Templates** for the files containing the parameters for training, classification and evaluation, all of them initializing the parameters by default values. There are four such files (one per subsystem and one for cross validation), produced for the user's convenience. These files can be edited by the user to change parameter values according to his or her wishes.

### 2.1.2 train_CNN_Classifier

The goal of this function is to train a classifier that can be used for the classification of images of silk fabrics. This function also delivers one or two log files that store information about the training and validation process. An input file *TrainingConfiguration.txt* containing the control parameters must exist in the current working directory. The following parameters need to be defined in the *TrainingConfiguration.txt* file:

- **cnn_config_file_name**: Name of the CNN configuration file generated by 'CreateDataset'. This file has to exist in directory 'cnn_config_file_path'.
- **cnn_config_file_path**: Path to the CNN configuration file.

- **master_file_name**: Name of the master file that lists the collection files with the available samples. This file must exist in the directory defined in *master_dir*.
- **master_dir**: Path to the master file.
- **logpath**: Path to where the trained model and the log files will be saved.
- **train_batch_size**: Number of samples that is used during each training iteration.
- **how_many_training_steps**: Number of training iterations.
- **learning_rate**: Learning rate for the training procedure.
- **validation_percentage**: Percentage of training samples that is used for validation.
- **how_often_validation**: Number of training iterations between two computations of the validation error.
- **weight_decay**: Weight of the regularization term in the loss function.
- **num_finetune_layers**: Number of layers of ResNet 152 to be fine-tuned during training.
- **evaluation_index**: Index of the collection file that will not be used for training the classifier.
- **flip_left_right**: Data augmentation: should horizontal flips be used (True) or not (False)?
- **flip_up_down**: Data augmentation: should vertical flips be used (True) or not (False)?
- **random_rotation90**: Data augmentation: should rotations by 90° be used (True) or not (False)?
- **gaussian_noise**: Data augmentation: Standard deviation of the Gaussian noise.

The input file for the function train_CNN_Classifier has to be formatted in the same way as the input file for the 'createDataset'. A template containing default values is generated by the function 'createDataset' and can be found at */silknow_image_classification/TrainingConfiguration.txt*. The output of the function is two-fold:

- A pre-trained model that can be used by the classification program to initialize the CNN in memory and to apply it for classification. It consists of three binary files in an internal format defined by tensorflow.
- One or two log files that store information about the training progress of the network. In all cases, there will be a log file showing the evolution of the training loss and other pieces of information; if validation is applied during training, a second log file will be created with meta information about the validation progress. Those files can be read using Tensorboard, a logging toolbox from Tensorflow.

### 2.1.3 apply_CNN_Classifier

The goal of this function is to classify images using a pre-trained model. It delivers the most probable classes for all variables and each image as well as the classification scores. An input file *ClassificationConfiguration.txt* containing the control parameters must exist in the current working directory. The following parameters need to be defined in the *ClassificationConfiguration.txt* file:

- **cnn_config_file_name**: Name of the CNN configuration file generated by 'CreateDataset'. This file has to exist in directory 'cnn_config_file_path'.
- **cnn_config_file_path**: Path to the CNN configuration file.
- **classification_master_file_name**: Name of a file that references the images to be classified (see main text). The file must exist in directory classification_master_dir .
- **classification_master_dir**: Path to the file referencing the images to be classified.
- **logpath**: Path to where the trained model and the log files will be saved.
- **classification_result_path**: Path to the directory where the files containing the classification results will be stored.

The input file defining the control parameters for this function has to be formatted like the input file for the function train_CNN_Classifier. A template containing default values is generated by the function ‘createDataset’ and can be found at */silknow_image_classification/ClassificationConfiguration.txt*. The output of the classification consists of two files:

- A text file containing the classification result, i.e. the most probable classes for each image. The first row of the file is a header line, starting with “#image_file”, followed by the names of the variables. All variable names have a preceding “#” and are separated by a tabulator. This line is followed by the classification results with one row per image to be classified. Each line starts with the relative path to the image, followed by the names of the most probable classes per variable in the order of variables indicated in the header line. Again, the class labels are separated by a tabulator.
- A text file containing the classification scores, i.e. the beliefs computed by the CNN for every class of every variable. This file contains one block per image to be classified. Each block starts with a header line containing the relative path to the input image. Each of the following lines in a block lists lists the CNN’s belief for all classes for one variable. A line first contains the name of the variable and then all class names with the corresponding beliefs in [%]. Blocks related to different images are separated by several blank lines.

### 2.1.4 evaluate_CNN_Classifier

The goal of this function is to evaluate an existing model with samples for which the class labels are already known. This function delivers the evaluation metrics as well as the confusion matrices for every variable; those results are introduced and explained in the SILKNOW Deliverable D4.4. The function also outputs the classification results for every test sample. Note that in order for the function to produce realistic results, the pre-trained network should only be evaluated using samples that were not used during training. The following parameters need to be defined in the *EvaluationConfiguration.txt* file:

- **cnn_config_file_name**: Name of the CNN configuration file generated by ‘CreateDataset’. This file has to exist in directory ‘cnn_config_file_path’.
- **cnn_config_file_path**: Path to the CNN configuration file.
- **logpath**: Path to where the trained model and the log files will be saved.
- **evaluation_master_file_name**: Name of the master file that gives access to the data for evaluation (see main text). It has to exist in the directory *evaluation_master_dir*.
- **evaluation_master_dir**: Path to the evaluation master file.
- **evaluation_result_path**: Path to the directory where the evaluation results will be saved.

The file defined by *evaluation_master_file_name* is a master file that just contains one row giving the name of a collection file. This file has to exist in the same directory as the evaluation master file. Similar to the other collection files, it lists the images that are to be used for evaluation along with the annotations; the format is identical to the one for master and collection files describing training samples. Templates for these files will be created automatically by the function ‘createDataset’. The input file defining the control parameters for this function has to be formatted like the input file for the function train_CNN_Classifier. A template containing default values is generated by function createDataset and can be found at */silknow_image_classification/EvaluationConfiguration.txt*. The output consists of the classification results for the evaluation samples as well as three files per variable:

- A text file named *Testing_VARIABLE_evaluation_results.txt* that lists the overall accuracy, the average F1 scores and the class-specific evaluation metrics for the variable named *VARIABLE*. First, the file reports the number of samples that were used for the evaluation for that variable and the overall accuracy. Note that the number of samples might be smaller than the total number of samples used for the evaluation, because incomplete samples will only contribute to the evaluation of some variables. The following lines contain the precision, recall and F1-score values for the individual classes, as well as the number of how often each class occurred in the evaluation samples. The last line of the file shows the average precision, recall and F1-score values. The average F1-score is generally a reliable indicator for assessing the quality of the prediction of the CNN represented by the imported pre-trained model.

- An image *Testing_VARIABLE_Confusion_Matrix.jpg* showing the confusion matrix C for the evaluation results for the variable named *VARIABLE*. The numbers in the matrix are unnormalized, i.e., they show absolute numbers of samples; he second output file is an image named *Testing_VARIABLE_Confusion_Matrix.jpg* and shows the confusion matrix for the evaluation of a single variable. To interpret the matrix more easily, the fields of the matrix are coloured, where a darker colour stands for a larger number.
- An image *Testing_VARIABLE_Confusion_Matrix_normalized.jpg* showing the normalized confusion matrix for the evaluation result for the variable named *VARIABLE*. The matrix is normalized by row, i.e., by the number of samples of a class according to the reference. Thus, an element on the main diagonal (corresponding to the correct predictions) is identical to the recall for the corresponding class.

### 2.1.5 crossvalidate_CNN_Classifier

The goal of this cross validation is to assess the classification performance of the classification model. As this function is in principle a combination of the training and evaluation functions, its outputs are a five trained models and logging files for training and validation as well as the evaluation results for every cross validation iteration, and finally the averaged evaluation results of all cross validation iterations. The following parameters need to be defined in the *CrossvalidationConfiguration.txt* file:

- **cnn_config_file_name**: Name of the CNN configuration file generated by 'CreateDataset'. This file has to exist in directory 'cnn_config_file_path'.
- **cnn_config_file_path**: Path to the CNN configuration file.
- **master_file_name**: Name of the master file that lists the collection files with the available samples. This file must exist in the directory defined in *master_dir*.
- **master_dir**: Path to the master file.
- **logpath**: Path to where the trained model and the log files will be saved.
- **evaluation_result_path**: Path to the directory where the evaluation results will be saved.
- **train_batch_size**: Number of samples that is used during each training iteration.
- **how_many_training_steps**: Number of training iterations.
- **learning_rate**: Learning rate for the training procedure.
- **validation_percentage**: Percentage of training samples that is used for validation.
- **how_often_validation**: Number of training iterations between two computations of the validation error.
- **weight_decay**: Weight of the regularization term in the loss function.
- **num_finetune_layers**: Number of layers of ResNet 152 to be fine-tuned during training.
- **evaluation_index**: Index of the collection file that will not be used for training the classifier.
- **flip_left_right**: Data augmentation: should horizontal flips be used (True) or not (False)?
- **flip_up_down**: Data augmentation: should vertical flips be used (True) or not (False)?
- **random_rotation90**: Data augmentation: should rotations by 90° be used (True) or not (False)?
- **gaussian_noise**: Data augmentation: Standard deviation of the Gaussian noise.

The list of control parameters is basically the same as for the training function. The only differences are that the cross validation function takes one more path, i.e. the path to where the evaluation results will be stored, and has one hyperparameter less, namely the *evaluation_index*, because all collection files will be used for the evaluation once over the course of the cross validation. Note that the default control parameter file template created by the function createDataset can be used for the training function as well as for the cross validation function, because the entry for *evaluation_index* will be ignored by the latter. The input file defining the control parameters for this function has to be formatted like the input file for the function train_CNN_Classifier. An exemplary input file can be found at */silknow_image_classification/CrossvalidationConfiguration.txt*.

## 2.2 Functions of the silknow_image_classification Toolbox

Created on Fri Nov 29 18:37:39 2019

@author: clermont

`silknow_image_classification.`**`apply_CNN_Classifier`**`(`*configfile*`)`

Applies the trained classifier to new data.

**Arguments:**

**configfile (*string*):** This variable is a string and contains the name of the configfile. All relevant information for applying the trained classifier is in this file. The configfile has to be stored in the same location as the script executing the classification function.

**Returns** No returns. The classification result will be written automatically into result file in the master direction. The name of this file can be chosen by the user in the control file.

`silknow_image_classification.`**`createDataset`**`(`*configfile*`)`

Creates the dataset for the CNN.

**Arguments:**

**configfile (*string*):** This variable is a string and contains the name of the configfile. All relevant information for the dataset creation are in this file. The configfile has to be stored in the same location as the script executing the classification function.

**Returns:** No returns. The classification result will be written automatically into result file in the master direction. The name of this file can be chosen by the user in the control file.

`silknow_image_classification.`**`crossvalidate_CNN_Classifier`**`(`*configfile*`)`

Carries out training of the CNN with cross validation.

**Arguments:**

**configfile (*string*):** This variable is a string and contains the name of the configfile. All relevant information for the dataset creation are in this file. The configfile has to be stored in the same location as the script executing the classification function.

**Returns:** No returns. The results of the evaluation are stored automatically in the directory given in the configfile.

`silknow_image_classification.`**`evaluate_CNN_Classifier`**`(`*configfile*`)`

Evaluates a pre-trained CNN.

**Arguments:**

**configfile (*string*):** This variable is a string and contains the name of the configfile. All relevant information for the evaluation is in this file. The configfile has to be stored in the same location as the script executing the evaluation function.

**Returns:** No returns. The results of the evaluation are stored automatically in the directory given in the configfile.

`silknow_image_classification.`**`train_CNN_Classifier`**(*configfile*)

Trains a classifier based on top of a pre-trained CNN.

**Arguments:**

**configfile (*string*):** This variable is a string and contains the name of the configfile. All relevant information for the training is in this file. The configfile has to be stored in the same location as the script executing the training function.

**Returns:** No returns. The trained graph (containing the tfhub_module and the trained classifier) is stored automatically in the directory given in the control file.

# PYTHON MODULE INDEX

# INDEX