Documentation of D4.5: SILKNOW Image Retrieval

Release 0.0.1

LUH

Jul 06, 2020

CONTENTS

CHAPTER

ONE

OVERVIEW OF THE DOCUMENTATION

This software provides Python functions for the retrieval of images and for training and evaluation of CNN-based retrieval models. It consists of six main parts:

1. creation of a dataset
2. training of a new retrieval model
3. creation of a descriptor index forming the search space for image retrieval
4. retrieval of images using an existing model and according index
5. evaluation of an existing retrieval model and the combined training
6. evaluation in a five-fold cross validation.

All functions take either configuration files or explicit parameter settings as an input and generally write their results in specified paths. The format required for the configuration files is described in the SILKNOW Deliverable D4.5.

The requirements for the silknow_image_retrieval toolbox are Python 3.6.4 and the following Python packages:

- numpy
- urllib3
- pandas (0.25.1)
- tqdm
- opencv-python
- tensorflow (1.13.1)
- tensorflow-hub (0.2.0)
- matplotlib
- sklearn
- scipy
- collections

CHAPTER

TWO

SILKNOW IMAGE RETRIEVAL

2.1 Summary of the silknow_image_retrieval Toolbox

The process flow of the image retrieval software and how it is embedded in the SILKNOW project is presented in Figure 1. Each depicted red reactangle in that figure belongs to a function of the software. All functions in the image retrieval package as well as a short description of them are listed in Table 1. Two versions are provided per function; one can be called by directly passing the respective parameters (`function_parameter`) and one getting the configuration file that contains all relevant parameter settings (`function_configfile`).

Name of the function	Short description of the function
`create_dataset_configfile`, `create_dataset_parameter`	Creates a dataset with samples and configuration files for training, index building, image retrieval and evaluation (Data Preparation).
`train_model_configfile`, `train_model_parameter`	Trains a new CNN on the basis of semantic similarity for descriptor estimations (Training).
`build_kDTree_configfile`, `build_kDTree_parameter`	Builds a spatial descriptor index out of provided feature vectors (Building Descriptor Index).
`get_kNN_configfile`, `get_kNN_parameter`	Provides the k nearest neighbours in the spatial descriptor index for a new feature vector (Get Nearest Neighbours).
`evaluate_model_configfile`, `evaluate_model_parameter`	Estimates evaluation metrics on the basis of a kNN- classification (Evaluation).
`crossvalidation_configfile`, `crossvalidation_parameter`	Performs 5-fold cross validation with training, index building, neighbour search an evaluation.

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

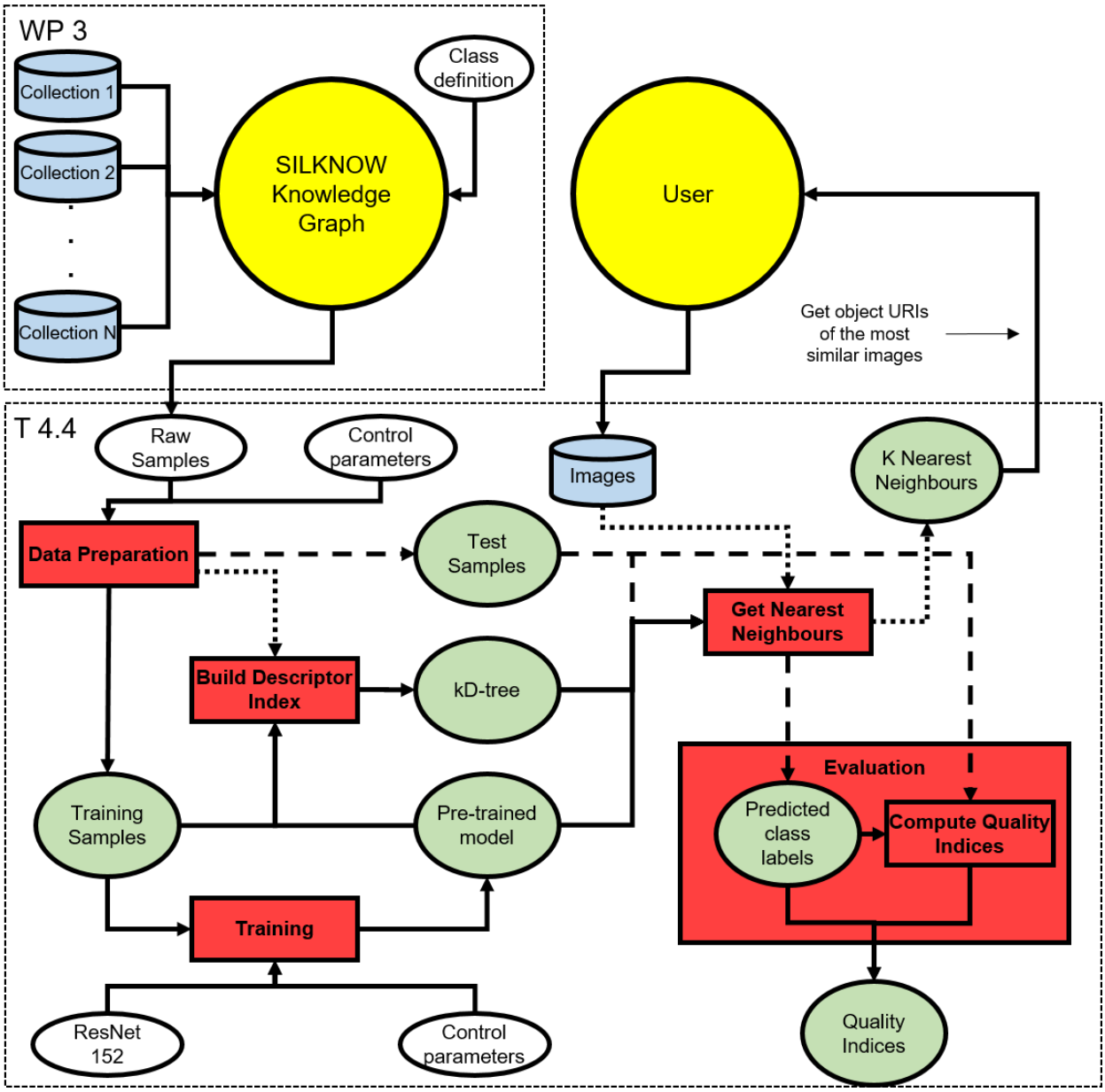

Fig. 1: Process flow of the SILKNOW image retrieval module and its embedding in the context of SILKNOW.

2.2 Functions of the silknow_image_retrieval Toolbox

Created on Mon Jan 20 14:07:11 2020

@author: clermont, dorozynski

silk_retrieval_functions.**build_kDTree_configfile**(*configfile*)

Builds a kD-Tree using a pre-trained network.

Arguments:

configfile (*string*): Path (including filename) to the configuration file defining the parameters for the function build_kDTree.

Returns: No returns. The "kdtree.npz"-file including the kD-tree is stored automatically in the directory given in the configuration file. The "kdtree.npz"-file is a dictionary containing the following key value pairs:

- Tree: The kD-tree accroding to the python toolbox sklearn.neighbors.
- Labels: The class label indices of the data whose feature vectors are stored in the tree nodes.
- DictTrain: The image data including their labels that was used to build the tree.
- relevant_variables: A list of the image data's labels that shall be considered.
- label2class_list: A list assigning the label indices to label names.

silk_retrieval_functions.**build_kDTree_parameter**(*model_path*, *master_file_tree*, *master_dir_tree*, *relevant_variables*, *savepath*)

Builds a kD-Tree using a pre-trained network.

Arguments:

model_path (*string*) Path (without filename) to a folder containing a pre-trained network. Only one pre-trained model should be stored in that model folder. It is exactly the path defined in the variable logpath in the function 'train_model_parameter'.

master_file_tree (*string*): The name of the master file that shall be used. The master file has to contain a list of the "collection.txt". All "collection.txt" files have to be in the same folder as the master file. The "collection.txt" files list samples with relative paths to the images and the according class labels. The paths in a "collection.txt" have to be "some_rel_path_from_location_of_master_txt/image_name.jpg". All samples from all stated collection files will be used to create the tree.

master_dir_tree (*string*): This variable is a string and contains the absolute path to the master file.

relevant_variables (*list*): A list containing the collection.txt's header terms of the labels to be considered. The terms must not contain blank spaces; use '_' instead of blank spaces!

- Example (string in configuration file): #timespan, #place
- Example (according list in code): [timespan, place]

savepath (*string*) Path to where the kD-tree "kdtree.npz" will be saved. It's a dictionary containing the following key value pairs:

- Tree: The kD-tree accroding to the python toolbox sklearn.neighbors.
- Labels: The class label indices of the data whose feature vectors are stored in the tree nodes.

- DictTrain: The image data including their labels that was used to build the tree.
- relevant_variables: A list of the image data's labels that shall be considered.
- label2class_list: A list assigning the label indices to label names.

Returns: No returns. The kD-tree is stored automatically in the directory given in the variable model_path.

silk_retrieval_functions.**create_dataset_configfile**(*configfile*)

Sets all dataset utilities up.

Arguments:

configfile (*string*): This variable is a string and contains the path (including filename) of the configuration file for data set creation. All relevant information for setting up the data set is in this file.

Returns: No returns. This function produces all files (data files as well as configuration files) needed for running the other functions in this python package.

silk_retrieval_functions.**create_dataset_parameter**(*csvfile, imgsavepath='../data/', minnumsamples=150, retaincollections=['garin', 'imatex', 'joconde', 'mad', 'mfa', 'risd'], num_labeled=1*)

Sets all dataset utilities up.

Arguments:

csvfile (*string*): Filename (including the path) of the .csv file containing information (like the deeplink) about the data in the SILKNOW knowledge graph.

imgsavepath (*string*): The path (without filename) that will contain the downloaded images. The original images will be in the folder "img_unscaled" in that directory and the rescaled images will be in the folder "img". It has to be relative to the current working directory.

minnumsaples (*int*): The minimum number of samples that should occur for a single class. Classes with fewer samples will not be considered.

retaincollections (*list*): A list containing the museums/collections in the knowledge graph that shall be considered for the data set creation. Data from museums/collections not stated in this list will be omitted. Possible values in the list according to EURECOM's export from the SILKNOW knowledge graph are: cer, garin, imatex, joconde, mad, met, mfa, mtmad, risd, unipa, vam.

num_labeled (*int*): Variable that indicates how many labels per sample should be available so that a sample is a valid sample and thus, part of the created dataset. The maximum number of num_labeled is 5, as five semantic variables are considered in the current implementation of this function. Choosing the maximum number means that only complete samples form the dataset. The value of num_labeled must not be smaller than 0.

Returns: No returns. This function produces all files (data files as well as configuration files) needed for running the other functions in this python package.

silk_retrieval_functions.**crossvalidation_configfile**(*configfile*)

Performs 5-fold crossvalidation.

Arguments:

configfile (*string*): Path (including filename) to the configuration file defining the parameters for the function crossvalidation.

Returns: No returns. The training information is stored automatically in the directory given in the configuration file.

silk_retrieval_functions.**crossvalidation_parameter**(*masterfile_name_crossval*, *masterfile_dir_crossval*, *logpath*, *train_batch_size*, *how_many_training_steps*, *learning_rate*, *add_fc*, *hub_num_retrain*, *loss_ind*, *relevant_variables*, *how_often_validation*, *val_percentage*, *num_neighbors*, *random_crop*, *random_rotation90*, *gaussian_noise*, *flip_left_right*, *flip_up_down*)

Performs 5-fold crossvalidation.

Arguments:

masterfile_name_crossval (*string*): The name of the master file that shall be used. The master file has to contain a list of the "collection.txt". All "collection.txt" files have to be in the same folder as the master file. The "collection.txt" files list samples with relative paths to the images and the according class labels. The paths in a "collection.txt" have to be "some_rel_path_from_location_of_master_txt/image_name.jpg". 80% of the data will be used for training (and validation according to the variabel val_percentage) and the remaining 20% for testing. The test set rotates in each crossvalidation iteration such that all provided samples are used for testing exactly once after crossvalidation is completed.

masterfile_dir_crossval (*string*): This variable is a string and contains the absolute path to the master file.

logpath (*string*) The path where all summarized training information and the trained network will be stored.

train_batch_size (*int*): This variable defines how many images shall be used for the classifier's training for one training step.

how_many_training_steps (*int*): Number of training iterations.

learning_rate (*float*): Specifies the learning rate of the Optimizer.

add_fc (*array of int*): The number of fully connected layers that shall be trained on top of the tensorflow hub Module is equal to the number of entries in the array. Each entry is an int specifying the number of nodes in the individual fully connected layers. If [1000, 100] is given, two fc layers will be added, where the first has 1000 nodes and the second has 100 nodes. If no layers should be added, an empty array '[]' has to be given.

hub_num_retrain (*int*): The number of layers from the tensorflow hub Module that shall be retrained.

loss_ind (*string*): The loss function that shall be utilized to estimate the network performance during training. Possible options:

- 'soft_contrastive': a contrastive loss with multi-label based similarity (complete samples)

- 'soft_contrastive_incomp_loss': a contrastive loss with multi-label based similarity (incomplete samples)
- 'soft_triplet': a contrastive loss with multi-label based similarity (complete samples)
- 'soft_triplet_incomp_loss': a contrastive loss with multi-label based similarity (incomplete samples)

relevant_variables (*list*): A list containing the collection.txt's header terms of the labels to be considered. The terms must not contain blank spaces; use '_' instead of blank spaces!

- Example (string in configuration file): #timespan, #place
- Example (according list in code): [timespan, place]

how_often_validation (*int*): Number of training iterations between validations.

val_percentage (*int*): Percentage of training data that will be used for validation.

num_neighbors (*int*): Number of closest neighbours that are retrieved from a kD-Tree during evaluation.

random_crop (*list*): Range of float fractions for centrally cropping the image. The crop fraction is drawn out of the provided range [lower bound, upper bound], i.e. the first and second values of random_crop. If [0.8, 0.9] is given, a crop fraction of e.g. 0.85 is drawn meaning that the crop for an image with the dimensions 200 x 400 pixels consists of the 170 x 340 central pixels.

random_rotation90 (*bool*): Data augmentation: should rotations by 90° be used (True) or not (False)?

gaussian_noise (*float*): Data augmentation: Standard deviation of the Gaussian noise

flip_left_right (*bool*): Data augmentation: should horizontal flips be used (True) or not (False)?

flip_up_down (*bool*): Data augmentation: should vertical flips be used (True) or not (False)?.

Returns: No returns. The training information is stored automatically in the directory given in the variable logpath.

`silk_retrieval_functions.`**`evaluate_model_configfile`**(*configfile*)

Evaluates a pre-trained model.

Arguments:

configfile (*string*): Path (including filename) to the configuration file defining the parameters for the function evaluate_model.

Returns: No returns. The results are stored automatically in the directory given in the configuration file. The evaluation results contain per evaluated relevant label:

- "evaluation_results.txt" containing quality metrics.
- "Confusion_Matrix.png": containing the confusion matrix with asolute numbers.
- "Confusion_Matrix_normalized.png": containing the row-wise normalized confusion matrix.

`silk_retrieval_functions.`**`evaluate_model_parameter`**(*pred_gt_path*, *result_path*)

Evaluates a pre-trained model.

Arguments:

pred_gt_path (*string*): Path (without filename) to a "pred_gt.npz" file that was produced by the function get_KNN. The folder should contain only one such file. The file contains the kD-tree as well as the labeled test data and the labels of the k nearest neighbours.

result_path (*string*): Path to where the evaluation results will be saved. The evaluation results contain per evaluated relevant label:

- "evaluation_results.txt" containing quality metrics.
- "Confusion_Matrix.png": containing the confusion matrix with asolute numbers.
- "Confusion_Matrix_normalized.png": containing the row-wise normalized confusion matrix.

Returns: No returns. The results are stored automatically in the directory given in the variable result_path.

silk_retrieval_functions.**get_kNN_configfile**(*configfile*)

Retrieves the k nearest neighbours from a given kdTree.

Arguments:

configfile (*string*): Path (including filename) to the configuration file defining the parameters for the function get_kNN.

Returns: No returns. The result is stored automatically in the directory given in the configuration file. In case of provided labeled data it's a dictionary "pred_gt.npz" containing the following key value pairs:

- Groundtruth: The label indicies of the provided data.
- Predictions: The label indices of the found k nearest neighbours.
- label2class_list: A list assigning the label indices to label names.

as well as text file "knn_list.txt" containing the predicted and ground truth labels. In case of provided unlabeled data it's a text file "knn_list.txt" containing the predictions. Furthermore, a CSV-file is created in any case containing the values

- input_image_name: The filename of the input image (including the path).
- kNN_image_names: The full image names of the kNN.
- kNN_kg_object_uri: The knowledge graph object URIs of the kNNs.
- kNN_kD_index: The indices of the kNN of the input image in the kD-tree.
- kNN_descriptor_dist: The distances of the descriptors of the kNN to the descriptor of the input image.

silk_retrieval_functions.**get_kNN_parameter**(*treepath*, *master_file_prediction*, *master_dir_prediction*, *bool_labeled_input*, *model_path*, *num_neighbors*, *savepath*)

Retrieves the k nearest neighbours from a given kdTree.

Arguments:

treepath (*string*): Path (without filename) to a "kdtree.npz"-file that was produced by the function build_kDTree. Only one kD-tree should be stored in that folder. It refers to the variable savepath in the function 'build_kDTree_parameter'. The provided kD-tree has to be built on the basis of the same pre-trained model as provided via the variable model_path of this function.

master_file_prediction (*string*): The name of the master file that shall be used. The master file has to contain a list of the "collection.txt". All "collection.txt" files have to be in the same folder as the master file. The "collection.txt" files list samples with relative paths to the images and the according class labels. The paths in a "collection.txt" have to be "some_rel_path_from_location_of_master_txt/image_name.jpg". For all samples from all stated collection files the feature vectors will be estimated resulting from the pre-trained model in model_path and afterwards the k nearest neighbours, where k refers to num_neighbors, will be found in the provided kD-tree.

master_dir_prediction (*string*): This variable is a string and contains the absolute path to the master file.

bool_labeled_input (*bool*): A boolean that states whether labels are available for the input image data (Tue) or not(False).

model_path (*string*) Path (without filename) to a pre-trained network. Only one pre-trained model should be stored in that model folder. It refers to the variable logpath in the function 'train_model_parameter'.

num_neighbors (*int*): Number of closest neighbours that are retrieved from a kD-Tree during evaluation.

savepath (*string*) Path to where the results will be saved. In case of provided labeled data it's a dictionary "pred_gt.npz" containing the following key value pairs:

- Groundtruth: The label indicies of the provided data.
- Predictions: The label indices of the found k nearest neighbours.
- label2class_list: A list assigning the label indices to label names.

as well as text file "knn_list.txt" containing the predicted and ground truth labels. In case of provided unlabeled data it's a text file "knn_list.txt" containing only the predictions. Furthermore, a CSV-file "kNN_LUT.csv" is created in any case containing the values

- input_image_name: The filename of the input image (including the path).
- kNN_image_names: The full image names of the kNN.
- kNN_kg_object_uri: The knowledge graph object URIs of the kNNs.
- kNN_kD_index: The indices of the kNN of the input image in the kD-tree.
- kNN_descriptor_dist: The distances of the descriptors of the kNN to the descriptor of the input image.

Returns: No returns. The result is stored automatically in the directory given in the variable savepath.

`silk_retrieval_functions.`**`train_model_configfile`**(*configfile*)

Trains a new model for calculating feature vectors.

Arguments:

configfile (*string*): This variable is a string and contains the path (including filename) of the configuration file. All relevant information for the training is in this file.

Returns: No returns. The trained graph (containing the pre-trained ResNet 152 and the trained new layers) is stored automatically in the directory given in the configuration file.

silk_retrieval_functions.**train_model_parameter**(*master_file*, *master_dir*, *logpath*, *train_batch_size*, *how_many_training_steps*, *learning_rate*, *add_fc*, *hub_num_retrain*, *loss_ind*, *relevant_variables*, *how_often_validation*, *val_percentage*, *random_crop*, *random_rotation90*, *gaussian_noise*, *flip_left_right*, *flip_up_down*)

Trains a new model for calculating feature vectors.

Arguments:

master_file (*string*): The name of the master file that shall be used. The master file has to contain a list of the "collection.txt". All "collection.txt" files have to be in the same folder as the master file. The "collection.txt" files list samples with relative paths to the images and the according class labels. The paths in a "collection.txt" have to be "some_rel_path_from_location_of_master_txt/image_name.jpg". All samples from all stated collection files will be used for training and, possibly, validation.

master_dir (*string*): This variable is a string and contains the absolute path to the master file.

logpath (*string*) The path where all summarized training information and the trained network will be stored.

train_batch_size (*int*): This variable defines how many images shall be used for the classifier's training for one training step.

how_many_training_steps (*int*): Number of training iterations.

learning_rate (*float*): Specifies the learning rate of the Optimizer.

add_fc (*array of int*): The number of fully connected layers that shall be trained on top of the tensorflow hub Module is equal to the number of entries in the array. Each entry is an int specifying the number of nodes in the individual fully connected layers. If [1000, 100] is given, two fc layers will be added, where the first has 1000 nodes and the second has 100 nodes. If no layers should be added, an empty array '[]' has to be given.

hub_num_retrain (*int*): The number of layers from the tensorflow hub Module that shall be retrained.

loss_ind (*string*): The loss function that shall be utilized to estimate the network performance during training. Possible options:

- 'soft_contrastive': a contrastive loss with multi-label based similarity (complete samples)
- 'soft_contrastive_incomp_loss': a contrastive loss with multi-label based similarity (incomplete samples)
- 'soft_triplet': a contrastive loss with multi-label based similarity (complete samples)
- 'soft_triplet_incomp_loss': a contrastive loss with multi-label based similarity (incomplete samples)

relevant_variables (*list*): A list containing the collection.txt's header terms of the labels to be considered. The terms must not contain blank spaces; use '_' instead of blank spaces!

- Example (string in configuration file): #timespan, #place
- Example (according list in code): [timespan, place]

how_often_validation (*int*): Number of training iterations between validations.

val_percentage (*int*): Percentage of training data that will be used for validation.

random_crop (*list*): Range of float fractions for centrally cropping the image. The crop fraction is drawn out of the provided range [lower bound, upper bound], i.e. the first and second values of random_crop. If [0.8, 0.9] is given, a crop fraction of e.g. 0.85 is drawn meaning that the crop for an image with the dimensions 200 x 400 pixels consists of the 170 x 340 central pixels.

random_rotation90 (*bool*): Data augmentation: should rotations by 90° be used (True) or not (False)?

gaussian_noise (*float*): Data augmentation: Standard deviation of the Gaussian noise

flip_left_right (*bool*): Data augmentation: should horizontal flips be used (True) or not (False)?

flip_up_down (*bool*): Data augmentation: should vertical flips be used (True) or not (False)?.

Returns: No returns. The trained graph (containing the pre-trained ResNet 152 and the trained new layers) is stored automatically in the directory given in the variable logpath.

PYTHON MODULE INDEX

S

INDEX