Documentation of D4.6: SILKNOW Image Processing modules

Release 0.0.1

LUH

Apr 08, 2021

CONTENTS

CHAPTER

ONE

SILKNOW IMAGE CLASSIFICATION

1.1 Overview of the documentation

This toolbox (https://github.com/silknow/image-classification) provides python functions for the classification of images. It consists of four main parts: The creation of a dataset, the training of a new classifier, the evaluation of an existing classifier and the classification of images using an existing classifier. All functions take configuration files as an input and generally write save their results in specified paths. The format required for the configuration files is described in Deliverable D4.4.

The requirements for the silknow_image_classification toolbox are python 3.6 and the following python packages:

- urllib3
- numpy
- pandas==0.24.1
- tqdm
- opencv-python
- tensorflow-gpu==1.14.0
- tensorflow-hub==0.6.0
- matplotlib
- sklearn
- scipy
- collections
- xlsxwriter

1.2 Summary of the silknow_image_classification Toolbox

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

Name of the function	Short description of the function
create_dataset_from_csv_parameter	Creates a dataset with samples for the training and evaluation.
train_model_parameter	Trains a new classification model.
classify_images_parameter	Classifies images using an existing classification model.
evaluate_model_parameter	Evaluates an existing classification model.
crossvalidation_parameter	Trains and evaluates a new model using cross validation.

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

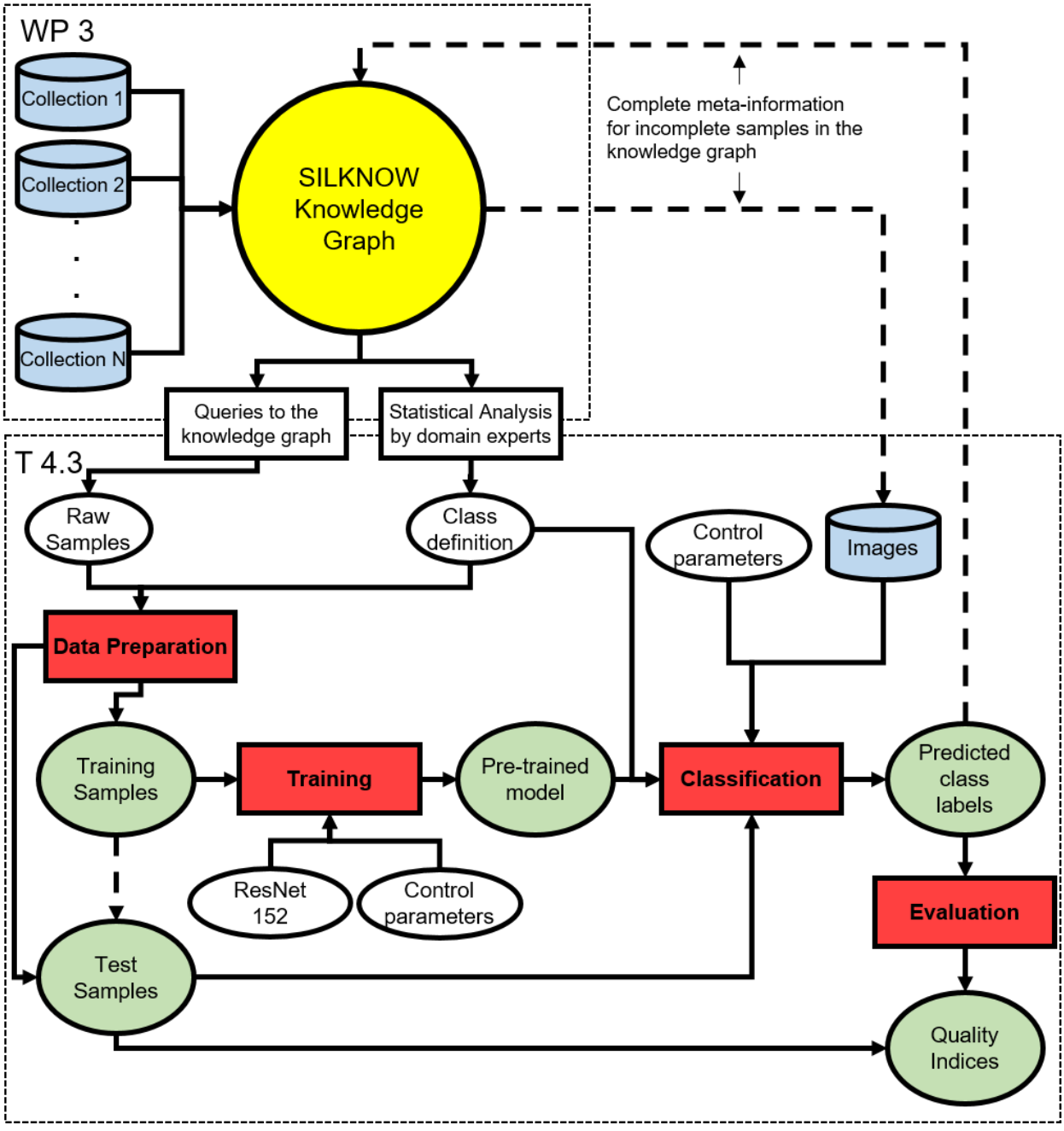

Fig. 1: Process flow of the SILKNOW image classification module and its embedding in the context of SILKNOW.

1.3 Functions of the silknow_image_classification Toolbox

silk_classification_func.**classify_images_parameter**(*masterfile_name*, *masterfile_dir*, *model_dir*, *result_dir*, *bool_unlabeled_dataset=None*, *image_based_samples=True*, *multi_label_variables=None*, *sigmoid_activation_thresh=0.5*)

Classifies images.

Arguments:

masterfile_name (*string*): Filename of the masterfile which states the collection files used for training and validation.

masterfile_dir (*string*): Directory where the masterfile is stored.

model_dir (*string*): Directory where the trained CNN will is stored. It is identical to log_dir in the training function.

result_dir (*string*): Directory where the results of the performed classification will be stored.

image_based_samples (bool): Has to be True if the collection files state image based samples (i.e. one image per sample). Has to be False if the collection files state record based samples (i.e. multiple images per sample).

multi_label_variables (*list of strings*): A list of variable names of the five semantic variables (relevant_variables) that have multiple class labels per variable to be used. A complete list would be ["material", "place", "timespan", "technique", "depiction"]. This list has to match the list in multi_label_variables that was used for the training of the selected model in model_dir.

sigmoid_activation_thresh (*float*): Only if multi_label_variables is not None. A float threshold defining the minimum value of the sigmoid activation in case of a multi-label classification that a class needs to have to be predicted.

Returns: No returns. This function produces all files needed for running the subsequent software.

silk_classification_func.**create_dataset_from_csv_parameter**(*csvfile*, *imgsavepath*, *master_file_dir*, *minnumsamples*, *retaincollections*, *num_labeled*, *multi_label_variables=None*)

Creates a dataset

Arguments:

csvfile (*string*): Path including the filename of the csv file containing the data exported from the SILKNOW knowledge graph (by EURECOM) or to data that is structured in the same way. An example is given in https://github.com/silknow/image-classification/tree/master/silknow_image_classification/samples.

imgsavepath (*string*): Path to were the images shall be downloaded or were the beforehand downloaded images are stored. All images in the csv file (rawCSVFile) that fulfill the further user-defined requirements are considered, i.e. images in that director that are not part of the csv file won't be considered. All images have to be in the folder; subfolders are not considered.

master_file_dir (*string*): Path where the created master file will be stored. The master file contains all created collection files. These collection files contain the relative paths from masterfileDirectory to the images in imageSaveDirectory as well as the images' annotations.

minnumsamples (*int*): The minimum number of images that shall contribute to a certain class. Classes (e.g. Italy) with less than minNumSamplesPerClass samples are not considered for the variable (e.g. place) with that class (e.g. Italy), i.e. the class label is set to 'nan' (unknown).

retaincollections (*list of strings*): A list of the names of the museums in the csv file that shall be considered for the dataset creation.

num_labeled (*int*): The minimum number of labels that shall be available for a sample. If a sample has less labels, it won't be part of the created dataset. This number of labels is counted after potentially setting labels to 'nan' (unknown) due to the restrictions in minNumSamplesPerClass. The maximum number of labels is 5 in the current software implementation, i.e. one label for each of the five semantic variables (relevant_variables in other functions of silknow_image_classification).

multi_label_variables (*list of strings*): A list of the SILKNOW knowledge graph names of the five semantic variables that have multiple class labels per variable to be used in subsequent functions. A complete list would be ["material_group", "place_country_code", "time_label", "technique_group", "depict_group"].

Returns: No returns. This function produces all files needed for running the subsequent software.

silk_classification_func.**crossvalidation_parameter**(*masterfile_name*, *masterfile_dir*, *log_dir*, *num_finetune_layers*, *relevant_variables*, *batchsize*, *how_many_training_steps*, *how_often_validation*, *validation_percentage*, *learning_rate*, *random_crop*, *random_rotation90*, *gaussian_noise*, *flip_left_right*, *dropout_rate*, *flip_up_down*, *nameOfLossFunction*, *multi_label_variables*, *sigmoid_activation_thresh*)

Perform 5-fold crossvalidation

Arguments:

masterfile_name (*string*): Filename of the masterfile which states the collection files used for training and validation.

masterfile_dir (*string*): Directory where the masterfile is stored.

log_dir (*string*): Directory where the trained CNN will be stored.

num_joint_fc_layer (int): Number of joint fully connected layers.

num_nodes_joint_fc (int): Number of nodes in each joint fully connected layer.

num_finetune_layers (int): Number of layers of the pretrained feature extraction network that will be finetuned.

relevant_variables (list): List of strings that defines the relevant variables.

batchsize (int): Number of samples per training iteration.

how_many_training_steps (int): Number of training iterations.

how_often_validation (int): Number of training iterations between validation steps.

validation_percentage (int): Percentage of training samples that will be used for validation.

learning_rate (float): Learning rate.

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)?.

image_based_samples (bool): Has to be True if the collection files state image based samples (i.e. one image per sample). Has to be False if the collection files state record based samples (i.e. multiple images per sample).

dropout_rate (float): Value between 0 and 1 that defines the probability of randomly dropping activations between the pretrained feature extractor and the fully connected layers.

nameOfLossFunction (bool): Indicates the loss function that shall be used. Available functions are listed in LossCollections.py.

multi_label_variables (*list of strings*): A list of variable names of the five semantic variables (relevant_variables) that have multiple class labels per variable to be used. A complete list would be ["material", "place", "timespan", "technique", "depiction"].

sigmoid_activation_thresh (*float*): Only if multi_label_variables is not None. A float threshold defining the minimum value of the sigmoid activation in case of a multi-label classification that a class needs to have to be predicted.

Returns: No returns.

`silk_classification_func.`**`evaluate_model_parameter`**(*pred_gt_dir*, *result_dir*, *multi_label_variables=None*)

Evaluates a model

Arguments:

pred_gt_dir (*string*): Directory containing the results of the classification function (result_dir in classify_images_parameter).

result_dir (*string*): Directory where the evaluation results will be stored.

multi_label_variables (*list of strings*): A list of variable names of the five semantic variables (relevant_variables) that have multiple class labels per variable to be used. A complete list would be ["material", "place", "timespan", "technique", "depiction"]. This list has to match the list in multi_label_variables that was used for the training of the selected model to be evaluated.

Returns: No returns.

silk_classification_func.**train_model_parameter**(*masterfile_name*, *masterfile_dir*, *log_dir*, *num_joint_fc_layer*, *num_nodes_joint_fc*, *num_finetune_layers*, *relevant_variables*, *batchsize*, *how_many_training_steps*, *how_often_validation*, *validation_percentage*, *learning_rate*, *weight_decay*, *num_task_stop_gradient*, *random_crop*, *random_rotation90*, *gaussian_noise*, *flip_left_right*, *flip_up_down*, *image_based_samples*, *dropout_rate*, *nameOfLossFunction*, *multi_label_variables*, *lossParameters={}*)

Trains a new classifier.

Arguments:

masterfile_name (*string*): Filename of the masterfile which states the collection files used for training and validation.

masterfile_dir (*string*): Directory where the masterfile is stored.

log_dir (*string*): Directory where the trained CNN will be stored.

num_joint_fc_layer (int): Number of joint fully connected layers.

num_nodes_joint_fc (int): Number of nodes in each joint fully connected layer.

num_finetune_layers (int): Number of layers of the pretrained feature extraction network that will be finetuned.

relevant_variables (list): List of strings that defines the relevant variables.

batchsize (int): Number of samples per training iteration.

how_many_training_steps (int): Number of training iterations.

how_often_validation (int): Number of training iterations between validation steps.

validation_percentage (int): Percentage of training samples that will be used for validation.

learning_rate (float): Learning rate.

weight_decay (float): Factor for the L2-loss of the weights.

num_task_stop_gradient (int): Samples with up to num_task_stop_gradient missing labels are used for training the joint layer. Samples with more missing labels will only be used for training the task-specific branches.

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)?.

image_based_samples (bool): Has to be True if the collection files state image based samples (i.e. one image per sample). Has to be False if the collection files state record based samples (i.e. multiple images per sample).

dropout_rate (float): Value between 0 and 1 that defines the probability of randomly dropping activations between the pretrained feature extractor and the fully connected layers.

nameOfLossFunction (bool):

Indicates the loss function that shall be used:

- If "sce": Softmax cross entropy loss for multi-task learning with incomplete samples.

(Note: both single-task learning and the complete samples case are special cases of "sce") - If "focal": Focal softmax cross entropy loss for multi-task learning with incomplete samples. (Note: both single-task learning and the complete samples case are special cases of "focal") - If "mixed_sce": Softmax cross entropy loss (for variables listed in relevant_variables, but not in multi_label_variables) combined with Sigmoid cross entropy loss (for variables listed both in relevant_variables and multi_label_variables) for multi-task learning with incomplete samples. (Note: both single-task learning and the complete samples case are special cases of "mixed_sce")

multi_label_variables (*list of strings*): A list of variable names of the five semantic variables (relevant_variables) that have multiple class labels per variable to be used. A complete list would be ["material", "place", "timespan", "technique", "depiction"].

lossParameters (bool): Indicates (optional) parameters for the chosen loss function. Specifications can be found in LossCollections.py.

Returns: No returns. This function produces all files needed for running the software.

CHAPTER

TWO

SILKNOW IMAGE RETRIEVAL

2.1 Overview of the documentation

This software (https://github.com/silknow/image-retrieval) 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

2.2 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 2: Names and short descriptions for the functions of the SILKNOW image retrieval software.

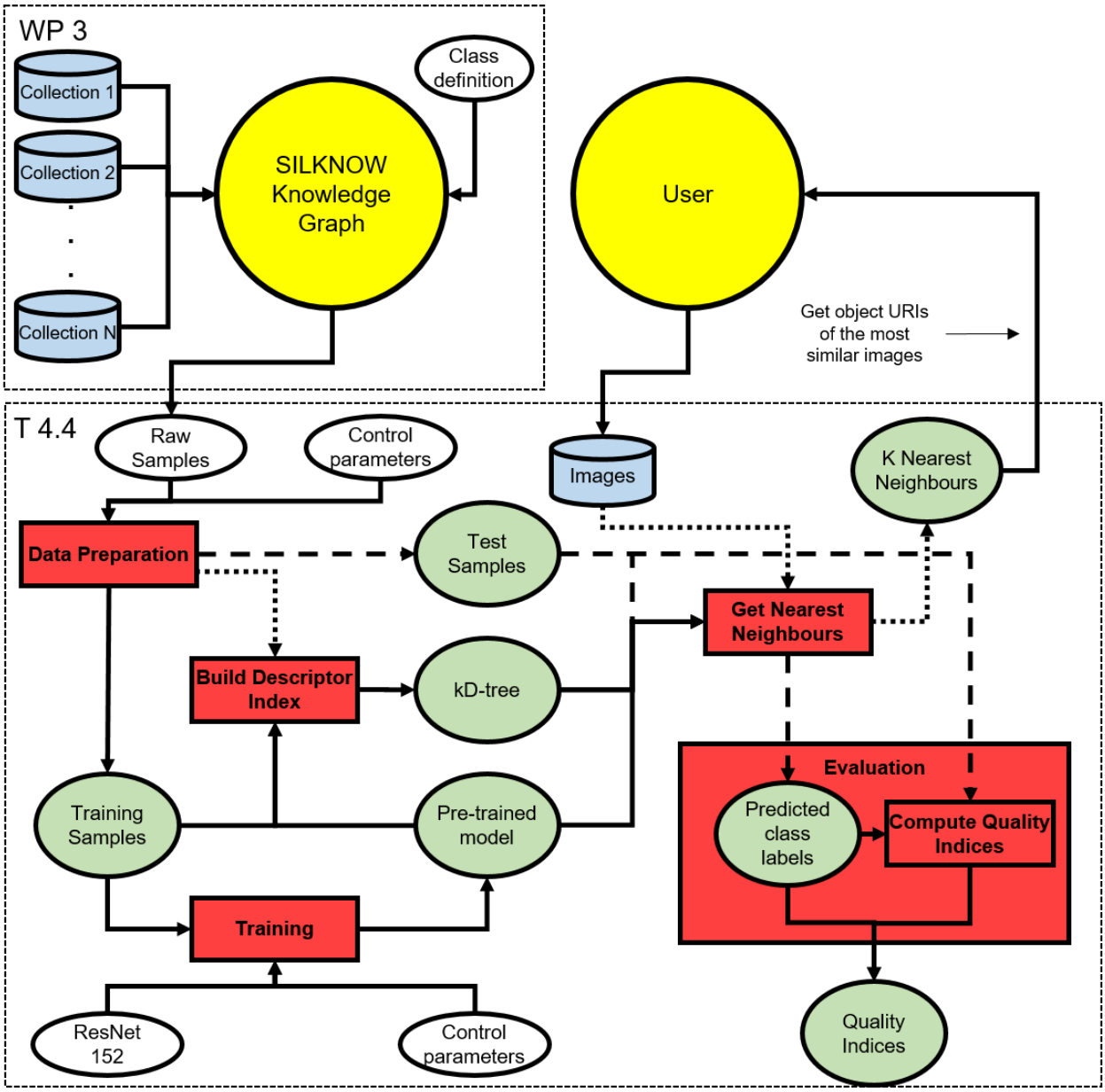

Fig. 1: Process flow of the SILKNOW image retrieval module and its embedding in the context of SILKNOW.

2.3 Functions of the silknow_image_retrieval Toolbox

silk_retrieval_func.**build_kDTree_parameter**(*model_dir*, *master_file_tree*, *master_dir_tree*, *tree_dir*, *relevant_variables*)

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

Arguments:

model_dir (*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 model_dir 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 (according list in code): [timespan, place]

tree_dir (*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 tree_dir.

silk_retrieval_func.**create_dataset_parameter**(*csvfile*, *imgsavepath*, *minnumsamples*, *retaincollections*, *num_labeled*, *master_file_dir*, *flagRuleDataset*, *masterfileRules*, *multi_label_variables=None*)

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', 'mobilier', 'mtmad', 'paris-musees', 'risd', 'smithsonian', 'unipa', 'vam', 'venezia', 'versailles'.

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.

master_file_dir (*string*): The path (without filename) that will contain the created master file listing the created collection files. The collection files contain the relative paths from the master_file_dir to the storage location of the images as well as the class labels assigned to the object (recrd in the SILKNOW knowledge graph) that the image depicts for the five semantic variables timespan, technique, place, material, depiction (according to the group level in the SILKNOW data model).

flagRuleDataset (*bool*): Boolean variable indicating whether additional data contained in rules formulated by the cultural heritage experts shall be considered in the dataset creation. If True, images belonging to objects with an object URI mentioned in a rule that are not yet part of the (labeled) dataset will be processed.

masterfileRules (*string*): Name of the rule master file that is assumed to be in master_file_dir. The rule master file lists one rule_file.txt per domain expert rule. Each such rule_file.txt contains one object uri per line that belongs to an object assigned to the respective rule.

multi_label_variables (*list of strings*): A list of the SILKNOW knowledge graph names of the five semantic variables that have multiple class labels per variable to be used in subsequent functions. A complete list would be ["material_group", "place_country_code", "time_label", "technique_group", "depict_group"].

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

silk_retrieval_func.**cross_validation_parameter**(*master_file_name*, *master_file_dir*, *master_file_rules_name*, *master_file_similar_obj*, *master_file_dissimilar_obj*, *log_dir*, *model_dir*, *tree_dir*, *pred_gt_dir*, *eval_result_dir*, *add_fc*, *num_fine_tune_layers*, *batch_size*, *semantic_batch_size_fraction*, *rules_batch_size_fraction*, *how_many_training_steps*, *learning_rate*, *validation_percentage*, *how_often_validation*, *loss_ind*, *num_neighbors*, *relevant_variables*, *variable_weights*, *loss_weight_semantic*, *loss_weight_rules*, *loss_weight_colour*, *loss_weight_augment*, *multi_label_variables*, *random_crop*, *random_rotation90*, *gaussian_noise*, *flip_left_right*, *flip_up_down*)

Performs 5-fold cross validation.

Arguments:

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

master_file_rules_name (*string*): The name of the master file containing the additional rules samples that shall be used. The master file has to contain a list of the "collection_rules.txt". All "collection_rules.txt" files have to be in the same folder as the master file. The "collection_rules.txt" files list samples with relative paths to the images and all class labels are set to nan (unknown) as they are irrelevant - all images with relevant class labels are contained in the collections listed in master_file_name. The paths in a "collection_rules.txt" have to be "some_rel_path_from_location_of_master_txt/image_name.jpg". All samples from all stated rule collection files will be used for training.

master_file_similar_obj (*string*): The name of the master file listing one file per rule that describe similar objects only. Each listed file contains one object URI per line of an object in the SILKNOW knowledge graph that contributes to the respective "similar rule".

master_file_similar_obj (*string*): The name of the master file listing one file per rule that describe dissimilar objects only. Each listed file contains one object URI per line of an object in the SILKNOW knowledge graph that contributes to the respective "dissimilar rule".

log_dir (*string*) The path where all summarized training information will be stored.

model_dir (*string*) Path (without filename) to a folder containing a pre-trained network. Only one pre-trained model should be stored in that model folder.

tree_dir (*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.

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

pred_gt_dir (*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.

eval_result_dir (*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.

batch_size (*int*): This variable defines how many images shall be used for the retrieval model'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.

num_fine_tune_layers (*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:

- 'triplet': a contrastive loss with multi-label multi-variable based semantic similarity
- 'colour': a colour distribution-based similarity
- 'rule': a cultural heritage domain expert rule-based similarity
- 'self_augment': similarity of the original image an an augmentation of that image
- 'combined_similarity_loss': combines all other similarities (exception: contrastive)

in one similarity definition.

loss_weight_semantic (*float*): Only, if loss_ind == combined_similarity_loss. The weight (float in [0, 1]) for the semantic similarity in the combined similarity. The sum of loss_weight_semantic, loss_weight_rules, loss_weight_colour and loss_weight_augment has to be one.

loss_weight_rules (*float*): Only, if loss_ind == combined_similarity_loss. The weight (float in [0, 1]) for the rules similarity in the combined similarity. The sum of loss_weight_semantic, loss_weight_rules, loss_weight_colour and loss_weight_augment has to be one.

loss_weight_colour (*float*): Only, if loss_ind == combined_similarity_loss. The weight (float in [0, 1]) for the colour similarity in the combined similarity. The sum of loss_weight_semantic, loss_weight_rules, loss_weight_colour and loss_weight_augment has to be one.

loss_weight_augment (*float*): Only, if loss_ind == combined_similarity_loss. The weight (float in [0, 1]) for the self-augmentation similarity in the combined similarity. The sum of loss_weight_semantic, loss_weight_rules, loss_weight_colour and loss_weight_augment has to be one.

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 (according list in code): [timespan, place]

variable_weights (*list of floats*): List of weights (floats) in [0, 1] describing the importance of the semantic variables in the definition of semantic similarity. The order has to be the same as in relevant_variables and the sum of weights has to be one.

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

validation_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)?.

multi_label_variables (*list of strings*): A list of variable names of the five semantic variables (relevant_variables) that have multiple class labels per variable to be used. A complete list would be ["material", "place", "timespan", "technique", "depiction"].

semantic_batch_size_fraction (*float*): A batch of the size batch_size will be drawn from two types of silk images. The first type contains images coming along with at least one class label for relevant_variables and the second type contains additional fully unlabeled images that contribute to the CH domain expert's rules. This variable defines the proportion of the labeled images in the batch. The sum of semantic_batch_size_fraction and rules_batch_size_fraction has to be one.

rules_batch_size_fraction (*float*): A batch of the size batch_size will be drawn from two types of silk images. The first type contains images coming along with at least one class label for relevant_variables and the second type contains additional fully unlabeled images that contribute to the CH domain expert's rules. This variable defines the proportion of the unlabeled images in the batch. The sum of semantic_batch_size_fraction and rules_batch_size_fraction has to be one.

Returns: No returns; all results will be stored automatically. The training summary in log_dir, the trained models of cross validation iterations in model_dir, the according kD-trees is tree_dir, the results of the image retrieval in pred_gt_dir and the results of the evalutaion in eval_result_dir.

`silk_retrieval_func.`**`evaluate_model_parameter`**(*pred_gt_dir*, *eval_result_dir*)

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.

eval_result_dir (*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 eval_result_dir.

silk_retrieval_func.**get_kNN_parameter**(*tree_dir*, *master_file_retrieval*, *master_dir_retrieval*, *model_dir*, *pred_gt_dir*, *num_neighbors*, *bool_labeled_input*)

Retrieves the k nearest neighbours from a given kdTree.

Arguments:

tree_dir (*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_retrieval (*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_retrieval (*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_dir (*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.

pred_gt_dir (*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 pred_gt_dir.

silk_retrieval_func.**train_model_parameter**(*master_file_name*, *master_file_dir*, *master_file_rules_name*, *master_file_similar_obj*, *master_file_dissimilar_obj*, *log_dir*, *model_dir*, *add_fc*, *num_fine_tune_layers*, *batch_size*, *semantic_batch_size_fraction*, *rules_batch_size_fraction*, *how_many_training_steps*, *learning_rate*, *validation_percentage*, *how_often_validation*, *loss_ind*, *loss_weight_semantic*, *loss_weight_rules*, *loss_weight_colour*, *loss_weight_augment*, *variable_weights*, *relevant_variables*, *random_crop*, *random_rotation90*, *gaussian_noise*, *flip_left_right*, *flip_up_down*, *multi_label_variables*)

Trains a new model for calculating feature vectors.

Arguments:

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

master_file_rules_name (*string*): The name of the master file containing the additional rules samples that shall be used. The master file has to contain a list of the "collection_rules.txt". All "collection_rules.txt" files have to be in the same folder as the master file. The "collection_rules.txt" files list samples with relative paths to the images and all class labels are set to nan (unknown) as they are irrelevant - all images with relevant class labels are contained in the collections listed in master_file_name. The paths in a "collection_rules.txt" have to be "some_rel_path_from_location_of_master_txt/image_name.jpg". All samples from all stated rule collection files will be used for training.

master_file_similar_obj (*string*): The name of the master file listing one file per rule that describe similar objects only. Each listed file contains one object URI per line of an object in the SILKNOW knowledge graph that contributes to the respective "similar rule".

master_file_similar_obj (*string*): The name of the master file listing one file per rule that describe dissimilar objects only. Each listed file contains one object URI per line of an object in the SILKNOW knowledge graph that contributes to the respective "dissimilar rule".

log_dir (*string*) The path where all summarized training information will be stored.

model_dir (*string*) Path (without filename) to a folder containing a pre-trained network. Only one pre-trained model should be stored in that model folder.

batch_size (*int*): This variable defines how many images shall be used for the retrieval model'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.

num_fine_tune_layers (*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:

- 'triplet': a contrastive loss with multi-label multi-variable based semantic similarity
- 'colour': a colour distribution-based similarity
- 'rule': a cultural heritage domain expert rule-based similarity
- 'self_augment': similarity of the original image an an augmentation of that image
- 'combined_similarity_loss': combines all other similarities (exception: contrastive)

in one similarity definition.

loss_weight_semantic (*float*): Only, if loss_ind == combined_similarity_loss. The weight (float in [0, 1]) for the semantic similarity in the combined similarity. The sum of loss_weight_semantic, loss_weight_rules, loss_weight_colour and loss_weight_augment has to be one.

loss_weight_rules (*float*): Only, if loss_ind == combined_similarity_loss. The weight (float in [0, 1]) for the rules similarity in the combined similarity. The sum of loss_weight_semantic, loss_weight_rules, loss_weight_colour and loss_weight_augment has to be one.

loss_weight_colour (*float*): Only, if loss_ind == combined_similarity_loss. The weight (float in [0, 1]) for the colour similarity in the combined similarity. The sum of loss_weight_semantic, loss_weight_rules, loss_weight_colour and loss_weight_augment has to be one.

loss_weight_augment (*float*): Only, if loss_ind == combined_similarity_loss. The weight (float in [0, 1]) for the self-augmentation similarity in the combined similarity. The sum of loss_weight_semantic, loss_weight_rules, loss_weight_colour and loss_weight_augment has to be one.

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]

variable_weights (*list of floats*): List of weights (floats) in [0, 1] describing the importance of the semantic variabels in the definition of semantic similarity. The order has to be the same as in relevant_variables and the sum of weights has to be one.

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)?.

multi_label_variables (*list of strings*): A list of variable names of the five semantic variables (relevant_variables) that have multiple class labels per variable to be used. A complete list would be [“material”, “place”, “timespan”, “technique”, “depiction”].

semantic_batch_size_fraction (*float*): A batch of the size batch_size will be drawn from two types of silk images. The first type contains images coming along with at least one class label for relevant_variables and the second type contains additional fully unlabeled images that contribute to the CH domain expert’s rules. This variable defines the proportion of the labeled images in the batch. The sum of semantic_batch_size_fraction and rules_batch_size_fraction has to be one.

rules_batch_size_fraction (*float*): A batch of the size batch_size will be drawn from two types of silk images. The first type contains images coming along with at least one class label for relevant_variables and the second type contains additional fully unlabeled images that contribute to the CH domain expert’s rules. This variable defines the proportion of the unlabeled images in the batch. The sum of semantic_batch_size_fraction and rules_batch_size_fraction has to be one.

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 model_dir and the according training summary in log_dir.

PYTHON MODULE INDEX

S

INDEX