A. Framework

A.1. Configuration Parameters

Parameter	Possible values	Default	Description
class	A PASCAL class	bicycle	Specifies the class to restrict the bounding boxes to
parts	$x \in \mathbb{N}\backslash 0$	4	Number of parts per window
clusters	$x \in \mathbb{N}\backslash 0$	1000	Number of k-Means clusters to use
integrals_scale _factor	$x \in \mathbb{R}, 0 < x \leq 1$	1	Specifies the percentage to which a integral image should be downsampled to
stream_max	$x \in \mathbb{N}\backslash 0$	100	Maximum amount of images used from a PASCAL stream
stream_name	Any string	query	Name of the PASCAL stream to read images from
codebook_type	double, single	double	Internal datatype of the codebooks
codebook _scales_count	$x \in \mathbb{N}\backslash 0$	3	Amount of scale ranges in which integral images will be split into
nonmax_type _min	true, false	true	Specifies if $\frac{\text{area}(A \cap B)}{\min(\text{area}(A), \text{area}(B))}$ or $\frac{\text{area}(A \cap B)}{\text{area}(A \cup B)}$ should be used for a non-maximum suppression
use_calibration	true, false	true	If true, a score calibration based on the query image is performed
features_per _roi	$x \in \mathbb{R}, 0 < x$	2	Amount of feature patches which have to exist per window axis
query_from _integral	true, false	false	If true, the query codebooks will be retrieved from the image database instead of extracted from the query image
default_query _file	An image id	2008 _004363	Used to automate the experiments instead of asking interactively for a query image

default _bounding _box	[x, y, width, height]	Extracted from the PASCAL dataset	The bounding box of the query image part
use_libsvm _classification	true, false	true	If true, the `svmpredict` function is used to classify the database codebooks. Otherwise a custom implementation is used
expand_bboxes	true, false	true	Expands the bounding boxes by $\frac{1}{2}$ of the average feature patch size
naiive_integral _backend	true, false	true	Specifies to use full integral images without any memory optimization techniques
window_margin	$x \in \mathbb{N}\backslash 0$	10	Amount of pixels each window is shifted during the sliding window generation
max_window _scales	$x \in \mathbb{N}\backslash 0$	10	Maximum amount of times a scaling is performed during the sliding window generation
min_window _size	$x \in \mathbb{R}, 0 < x \leq 1$	0.5	Required percentage of a window area which has to be placed over an image (windows are getting clipped at the image boundaries)
window _generation _relative_move	$x \in \mathbb{R}$	0	Percentage of its size a window should move during sliding window (0 means fixed window _margin pixels)
max_window _image_ratio	$x \in \mathbb{R}, 0 < x \leq 1$	1.0	Maximum allowed size of a window in relation to the image
use_kdtree	true, false	false	Enables the kd-Tree storage backend
integral _backend _overwrite	true, false	false	Enables the integral matrix reconstruction via overwriting the bottom right values
integral _backend _sum	true, false	false	Enables the integral matrix reconstruction via summing up the bottom right values
integral _backend _matlab_sparse	true, false	false	Enables the MATLAB® sparse storage backend

precalced_windows	true, false	false	Disables integral images and precalculates codebooks in a sliding window manner
inverse_search	true, false	false	Use the information of integral images to reduce the amount of windows generated.
memory_cache	true, false	true	Reuse already loaded databases and negative codebooks (useful for development)
use_threading	true, false	true	Enables the use of the parallel toolbox
window_prefilter	true, false	false	Prefilter the windows even further to reduce the amount of searched windows
fisher_backend	true, false	false	Use fisher vectors instead of k-means clusters (experimental as it is not practical)

Table A.1.: Configuration parameters for the part retrieval framework

A.2. API

A.2.1. DEMO

Demo implementation. Also creates entries in the results/queries folder

Syntax: `demo(train, ...)`

Inputs:

Name	**Description**
Name	Value pairs to override the default configuration
build	boolean to indicate the creation of an image database

Outputs:

Name	**Description**
results	The results of the query
num_windows	number of windows originally created to search through

A.2.2. GETIMAGEDB

Generate or load the image database

Syntax: `database = getImageDB(params, cluster_model)`

Inputs:

Name	Description
params	Configuration parameters
cluster_model	The model for codebook generation

Outputs:

Name	Description
database	The database of integral images

A.2.3. GETSVM

Train or load a exemplar SVM

Syntax: `svm_models = getSVM(params, cluster_model)`

Inputs:

Name	Description
cluster_model	The model to generate codebooks
parms	Configuration parameters

Outputs:

Name	Description
svm_models	SVM model struct

A.2.4. SEARCHINTERACTIVE

Do a complete interactive database searchAsks the user for a image in the dataset image path.Allows to mark a ROI to search forStores results in the results/queries subdirectories

Syntax: `searchInteractive(params, cluster_model)`

Inputs:

Name	Description
params	Configuration parameters
cluster_model	The model to generate codebooks

A.2.5. SEARCHDATABASE

Search the image database with the given SVM modelsStores results in the results/-queries subdirectories

Syntax: `results = searchDatabase(params, database, svm_models, fit_params, roi`

Inputs:

Name	Description
params	Configuration parameters
database	The image database as struct array of integral codebook images
roi	The region of interest [x_min, y_min, x_max, y_max]
fit_params	Gaussian curve 2D Vectors to adjust the SVM scores [μ, σ]
svm_models	Struct array of svm models

Outputs:

Name	Description
results	Cell array of results. Fields: curid, img, patch, score

A.2.6. SEARCHDATABASE

Search the image database with the given SVM modelsStores results in the results/-queries subdirectories

Syntax: `results = searchDatabase(params, database, svm_models, fit_params, roi`

Inputs:

Name	Description
params	Configuration parameters
database	The image database as struct array of integral codebook images
roi	The region of interest [x_min, y_min, x_max, y_max]
fit_params	Gaussian curve 2D Vectors to adjust the SVM scores [μ, σ]
svm_models	Struct array of svm models

Outputs:

Name	Description
results	Cell array of results. Fields: curid, img, patch, score

A.2.7. CALIBRATE_FIT

Estimates the gaussian parameters for the given svm models

Syntax: `fit_params = calibrate_fit(params, svm_models, query_file, cluster_model)`

Inputs:

Name	**Description**
params	Configuration parameters
cluster_model	The model to generate the codebooks
query_file	A pascal stream containing the file used to train the SVM
svm_models	The SVM models

Outputs:

Name	**Description**
fit_params	Nx2 matrix of gaussian parameters. N: number of SVM models, Gaussian Parameters: $[\mu, \sigma]$

A.2.8. CALIBRATE_RHO

Tries to readjust the rho of the given SVM models to obtain better scores

Syntax: `rhos = calibrate_rho(params, svm_models, query_file, cluster_model, ground_tr`

Inputs:

Name	**Description**
params	Configuration parameters
cluster_model	The model to generate the codebooks
ground_truths	Nx4 Matrix of bounding boxes inside the query image (e.g. the ROI)
query_file	A pascal stream containing the file used to train the SVM
svm_models	The SVM models

Outputs:

Name	**Description**
rhos	Mx1 vector of new rhos. M: number of SVM models

A.2.9. CONVERT_SERIALIZE

Creates a corresponding file with toggled serialization

Syntax: `convert_serialize(filename)`

Inputs:

Name	Description
filename	The file to invert

A.2.10. GET_DEFAULT_CONFIGURATION

returns a default configuration

Syntax: `params = get_default_configuration()`

Outputs:

Name	Description
params	The default configuration

A.2.11. GET_CODEBOOKS

Get integral codebooks from given features

Syntax: `codebooks = get_codebooks(params, features, cluster_model)`

Inputs:

Name	Description
params	Configuration struct
cluster_model	A model from get_cluster
features	A feature struct array. Required Fields: curid, X, bbs, window2feature

Outputs:

Name	Description
codebooks	A struct array with fields: I, curid, size

A.2.12. GET_CODEBOOK_INTEGRALS

Get integral codebooks from given features

Syntax: `integrals = get_codebook_integrals(params, features, cluster_model, rc`

Inputs:

Name	Description
params	Configuration struct
cluster_model	A model from get_cluster
features	A feature struct array. Required Fields: curid, X, bbs, I_size, scales
roi_size	Size of the query part

Outputs:

Name	Description
integrals	A struct array with fields: I, I_size, curid, scale_factor,max_size, min_size, tree,scores, idx, coords

A.2.13. GET_CACHE_BASEDIR

Gets and creates the cache dir

Syntax: `basedir = get_cache_basedir(params, create_dir)`

Inputs:

Name	Description
params	Configuration struct
create_dir	Boolean to indicate the automatic creation of the dir

Outputs:

Name	Description
basedir	The cache dir

A.2.14. GET_CACHE_NAME

Gets the cache filename

Syntax: `cachename = get_cache_name(params, roi_size, create_dir)`

Inputs:

Name	Description
params	Configuration struct
roi_size	Vector containing the size of the roi or empty
create_dir	Boolean to indicate the automatic creation of the dir

Outputs:

Name	Description
cachename	The cache file name

A.2.15. GET_IMG_CACHE_NAME

Gets the cache filename of a single image

Syntax: `imgcachename = get_img_cache_name(params, roi_size, create_dir)`

Inputs:

Name	Description
params	Configuration struct
roi_size	Vector containing the size of the roi or empty
create_dir	Boolean to indicate the automatic creation of the dir

Outputs:

Name	Description
imgcachename	The cache file name

A.2.16. CREATE_KD_TREE

Builds up a kd-Tree

Syntax: `tree = create_kd_tree(Is, remaining, scores, point_only)`

Inputs:

Name	Description
point_only	Boolean to indicate if scores should not be stored
scores	N scores
remaining	Logical index of N important points
Is	Integral image

A.2.17. GET_CLUSTER

Clusters the given features with kmeans

Syntax: `model = get_cluster( params, features )`

Inputs:

Name	Description
params	The configuration struct used for caching and profiling
features	The feature struct array (Required Fields: X)

Outputs:

Name	Description
model	The computed model (Fields: centroids. Methods: feature2codebook(params, feature), feature2codebookintegral(params, feature))

A.2.18. ADDFUNCTIONS

Adds function handles to the model

Syntax: `model = addfunctions(params, model)`

Inputs:

Name	Description
params	The configuration struct used for caching and profiling
model	The model to update

Outputs:

Name	Description
model	The updated model

A.2.19. FEATURE2CODEBOOK

Calculates a codebook from a given feature struct

Syntax: `codebook = feature2codebook(params, feature, model)`

Inputs:

Name	Description
params	The configuration struct. Required fields: parts, codebook_type, profile (if profiling is required)
model	The cluster model. Required fields: centroids
feature	The feature struct. Required fields: X, window2feature, bbs

Outputs:

Name	Description
codebook	A NxM matrix. N: params.parts * size(centroids, 1), M: length(window2feature)

A.2.20. FEATURE2CODEBOOK

Calculates a codebook integral from a given feature struct

Syntax: `codebook = feature2codebookintegral(params, feature, model)`

Inputs:

Name	Description
params	The configuration struct. Required fields: codebook_type, profile (if profiling is required)
model	The cluster model. Required fields: centroids
feature	The feature struct. Required fields: X, bbs, I_size, scales

Outputs:

Name	Description
scales	A Cell of size S containing the associated scales.
checkpoints	Always empty for this technique
codebook	A SxNxWxH matrix. S: different scales, N: size(centroids, 1), W: I_size(2), H: I_size(1)

A.2.21. FEATURE2CODEBOOK

Calculates a codebook integral from a given feature struct

Syntax: `codebook = feature2codebookintegral(params, feature, model)`

Inputs:

Name	Description
params	The configuration struct. Required fields: codebook_type, profile (if profiling is required)
model	The cluster model. Required fields: centroids
feature	The feature struct. Required fields: X, bbs, I_size, scales

Outputs:

Name	Description
scales	A Cell of size S containing the associated scales.
checkpoints	Locations of codebook assignments (logical)
codebook	A SxNxWxH matrix. S: different scales, N: size(centroids, 1), W: I_size(2), H: I_size(1)

A.2.22. FEATURE2CODEBOOK

Calculates a codebook from a given feature struct

Syntax: `codebook = feature2codebook(params, feature, model)`

Inputs:

Name	Description
params	The configuration struct. Required fields: parts, codebook_type, profile (if profiling is required)
model	The cluster model. Required fields: centroids
feature	The feature struct. Required fields: X, window2feature, bbs

Outputs:

Name	Description
codebook	A NxM matrix. N: params.parts * size(centroids, 1), M: length(window2feature)

A.2.23. FEATURE2CODEBOOK

Calculates a codebook integral from a given feature struct

Syntax: `codebook = feature2codebookintegral(params, feature, model)`

Inputs:

Name	Description
params	The configuration struct. Required fields: codebook_type, profile (if profiling is required)
model	The cluster model. Required fields: centroids
feature	The feature struct. Required fields: X, bbs, I_size, scales

Outputs:

Name	Description
scales	A Cell of size S containing the associated scales.
checkpoints	Always empty for this technique
codebook	A SxNxWxH matrix. S: different scales, N: size(centroids, 1), W: I_size(2), H: I_size(1)

A.2.24. FEATURE2CODEBOOK

Calculates a codebook integral from a given feature struct

Syntax: `codebook = feature2codebookintegral(params, feature, model)`

Inputs:

Name	Description
params	The configuration struct. Required fields: codebook_type, profile (if profiling is required)
model	The cluster model. Required fields: centroids
feature	The feature struct. Required fields: X, bbs, I_size, scales

Outputs:

Name	Description
scales	A Cell of size S containing the associated scales.
checkpoints	Always empty for this technique
codebook	A SxNxWxH matrix. S: different scales, N: size(centroids, 1), W: I_size(2), H: I_size(1)

A.2.25. GET_CODEBOOK_FROM_WINDOWS

Get images codebooks from given features

Syntax: `images = get_codebook_from_windows(params, features, cluster_model, rc`

Inputs:

Name	Description
params	Configuration struct
cluster_model	A model from get_cluster
features	A feature struct array. Required Fields: curid, X, bbs, I_size, scales
roi_size	Size of the query part

Outputs:

Name	Description
images	A struct array with fields: curid, scale_factor, max_size,min_size, bboxes, codebooks, images

A.2.26. GET_CACHE_BASEDIR

Gets and creates the cache dir

Syntax: `basedir = get_cache_basedir(params, create_dir)`

Inputs:

Name	Description
params	Configuration struct
create_dir	Boolean to indicate the automatic creation of the dir

Outputs:

Name	Description
basedir	The cache dir

A.2.27. GET_CACHE_NAME

Gets the cache filename

Syntax: `cachename = get_cache_name(params, roi_size, create_dir)`

Inputs:

Name	Description
params	Configuration struct
roi_size	Vector containing the size of the roi or empty
create_dir	Boolean to indicate the automatic creation of the dir

Outputs:

Name	Description
cachename	The cache file name

A.2.28. GET_IMG_CACHE_NAME

Gets the cache filename of a single image

Syntax: `imgcachename = get_img_cache_name(params, roi_size, create_dir)`

Inputs:

Name	Description
params	Configuration struct
roi_size	Vector containing the size of the roi or empty
create_dir	Boolean to indicate the automatic creation of the dir

Outputs:

Name	Description
imgcachename	The cache file name

A.2.29. GET_IMAGE

Loads a pascal image by its name

Syntax: `I = get_image( params, name )`

Inputs:

Name	Description
params	Configuration struct
name	The image name

Outputs:

Name	Description
I	The loaded image

A.2.30. GET_FEATURES_FROM_STREAM

Calculates HoG features from a given Pascal Stream

Syntax: `features = get_features_from_stream( params, stream )`

Inputs:

Name	Description
params	Configuration struct
stream	A Pascal stream

Outputs:

Name	Description
features	A feature struct array. Fields: curid, objectid,feature_type, I_size, X, M, scales, bbs, window2feature,area, all_scales

A.2.31. EXTRACT_QUERY_CODEBOOK

extracts a query codebook based on the configuration

Syntax: `query_codebooks = extract_query_codebook( params, cluster_model, query_file,`

Inputs:

Name	Description
params	Configuration struct
cluster_model	Model representing the current clustering method
query_file	Struct with information about the query file
roi_size	Bounding box of the query part

Outputs:

Name	Description
query_codebooks	the resulting codebook

A.2.32. PREPARE_FEATURES

load or generate features for the given stream set

Syntax: `features = prepare_features(params, query_stream_set)`

Inputs:

Name	Description
params	Configuration struct
query_stream_set	stream set containing the files to load

Outputs:

Name	Description
features	struct array of features

A.2.33. GET_FULL_NEG_MODEL

Loads the negative model for whitened HoGs

Syntax: `neg_model = get_full_neg_model()`

Outputs:

Name	Description
neg_model	The negative model

A.2.34. GET_SVMS

Get trained exemplar svms from given codebooks

Syntax: `svm_models = get_svms( params, query_codebooks, neg_codebooks )`

Inputs:

Name	**Description**
params	Configuration struct
query_codebooks	A codebook struct array. Required Fields: I, size, curid
neg_codebooks	Codebook struct array (Fields: I) or NxM matrix. N: Num codebooks, M: Codebook size

Outputs:

Name	**Description**
svm_models	Struct array of svm models. Fields: cb_size, codebook, curid, model

A.2.35. CLASSIFY_CODEBOOKS

Scores codebooks by a SVM model

Syntax: `scores = classify_codebooks(params, model, codebooks)`

Inputs:

Name	**Description**
params	Configuration
model	SVM Model
codebooks	Codebook struct array

Outputs:

Name	**Description**
scores	The score for each codebook

A.2.36. ADJUST_SCORES

adjusts scores based on given gaussian fit parametersScores will applied on a gaussian curve. The curve is shifted by $3*\sigma$and the resulting scores rescaled by $new\_scores * 2 - 1$

Syntax: `[ new_scores ] = adjust_scores( params, fit_params, scores )`

Inputs:

Name	Description
params	Configuration parameters as struct
fit_params	2D-Vector. [μ, σ]
scores	Score vector to adjust

Outputs:

Name	Description
new_scores	Adjusted scores

A.2.37. GET_RANGE_FOR_SCALE

get the range of given scales

Syntax: `[ min_size, max_size ] = get_range_for_scale(params, current_scales)`

Inputs:

Name	Description
params	Configuration struct
current_scales	NxM matrix of scales

Outputs:

Name	Description
max_size	vector with N maximum values
min_size	vector with N minimum values

A.2.38. FILTER_FEATURES

Filters given featuresRemoves features with too little texture or which are too close to the negative mode of whitened features

Syntax: `features = filter_features(params, features)`

Inputs:

Name	Description
params	The configuration struct. Used for profiling and caching
features	The feature struct array (Fields: X, M, distVec, scales, bbs, window2feature)

Outputs:

Name	Description
features	The filtered feature struct array with new logical vector deletedFeatures

A.2.39. CALC_CODEBOOKS

Extracts codebooks and bounding boxes from a given image database

Syntax: `[ bboxes, codebooks, images ] = calc_codebooks(params, database, windo`

Inputs:

Name	Description
params	The configuration parameters, currently only required for profiling
database	The image database as struct array with the fields I, curid and optionally scale_factor
windows_bb	A Nx4 matrix of bounding boxes to to extract (x_min, y_min, x_max, y_max)
num_parts	(Even) number of segments a window should be divided to

Outputs:

Name	Description
codebooks	A Nx(M*num_parts) matrix of M dimensional codebooks
bboxes	A Nx4 matrix of bounding boxes related to the extracted codebooks (x_min, y_min, x_max, y_max)
images	A 1xN dimensional index vector for assigning codebooks to images

A.2.40. EXPECTEDCODEBOOKS

Tries to estimate the amount of codebooks extractedInternally used to preallocate the memory for speed

Syntax: `[expectedCodebookCount, codebookSize] = expectedCodebooks(database, wi`

Inputs:

Name	Description
database	The image database as struct array with the field I
windows_bb	A Nx4 matrix of bounding boxes to to extract
num_parts	(Even) number of segments a window should be divided to

Outputs:

Name	Description
expectedCodebookCount	Estimated number of codebooks which will be extracted
codebookSize	Size of the resulting codebooks (M*num_parts)

A.2.41. IS_IN_SCALE

Tests if the requested sizes are within the a given scale

Syntax: `in_scale = is_in_scale(params, min_size, max_size, requested_size)`

Inputs:

Name	Description
params	Configuration struct
max_size	upper bound (or [] if lower bound is requested)
min_size	lower bound (or [] if upper bound is requested)

Outputs:

Name	Description
in_scale	logic vector

A.2.42. CALC_WINDOWS

Calculates the windows wich have to be extracted in a sliding window approach

Syntax: `windows_bb = calc_windows(params, w, h, cbw, cbh, I )`

Inputs:

Name	Description
params	Configuration struct
w	Width of the image
I	Optional test image to produce a intagral.avi file for visualization
cbw	Smallest width of a window
h	Height of the image
cbh	Smallest height of a window

Outputs:

Name	Description
windows_bb	Nx4 matrix of windows (x_min, y_min, x_max, y_max)

A.2.43. EXTRACT_CODEBOOKS

Extracts codebooks from the image database and expands the bounding boxes if necessary

Syntax: `[bboxes, codebooks, images, windows, num_orig_windows] = extract_codeb`

Inputs:

Name	Description
params	Configuration
database	Image database struct array
pos	Bounding box of query [xmin, ymin, xmax, ymax]
svm_models	Struct array of SVM models

Outputs:

Name	Description
codebooks	Codebooks of reduced windows as Nx(M*num_parts)
num_orig_windows	Number of unreduced windows
bboxes	Bounding boxes of reduced windows as Nx4
windows	Bounding boxes of unreduced windows
images	Vector of image numbers per window

A.2.44. FILTER_WINDOWS_BY_INVERSE_SEARCH

Filters windows by searching for relevant codebook entries

Syntax: `filtered_windows = filter_windows_by_inverse_search(params, integral,`

Inputs:

Name	Description
params	Configuratoion
integral	Integral image
svm_model	Model to extract relevant dimensions from
windows	List of windows

A.2.45. SPARSE_CODEBOOK

Matlab implementation of the reconstruction of a kd-Tree integral image

Syntax: `codebook = sparse_codebook(integral, x, y)`

Inputs:

Name	Description
integral	An integral image with kd-Tree storage backend
x, y	Coordinate of requested codebook

Outputs:

Name	Description
codebook	the reconstructed codebook

A.2.46. GETCODEBOOKSFROMINTEGRAL

Extract codebooks from a single integral image

Syntax: `codebooks = getCodebooksFromIntegral(params, integral_img, bboxes, num_parts)`

Inputs:

Name	Description
params	Configuration struct
bboxes	Mx4 Matrix of bounding boxes to extract
integral_img	SxNxWxH Matrix. S: scales, N: Codebook Size, W: Width, H: Height
num_parts	(Even) number of segments to divide a single window into

Outputs:

Name	Description
codebooks	N2xSxM Matrix. N2: N*num_parts

A.2.47. GENERATECLUSTER

Generate a cluster model

Syntax: `cluster_model = generateCluster(params)`

Inputs:

Name	Description
params	Configuration parameters

Outputs:

Name	Description
cluster_model	The model

A.2.48. LOAD_DATABASE

Loads the image database or generates it

Syntax: `database = load_database(params, cluster_model, roi_size)`

Inputs:

Name	Description
params	Configuration
cluster_model	The model to use for clustering
roi_size	Size of the query part

Outputs:

Name	Description
database	The loaded/generated database as struct array

A.2.49. ESTIMATE_FIT_PARAMS

Estimates parameters of a gaussian curveCan be used to adjust the scores

Syntax: `fit_params = estimate_fit_params( params, scores )`

Inputs:

Name	Description
params	Configuration struct (currently not used)
scores	Score vector to fit to

Outputs:

Name	Description
fit_params	2D-Vector of parameters [μ, σ]

A.2.50. GET_NEG_CODEBOOKS

Load the negative codebooks for SVM training

Syntax: `neg_codebooks = get_neg_codebooks(params, cluster_model)`

Inputs:

Name	Description
params	Configuration
cluster_model	The model used to cluster features if no cache available

Outputs:

Name	Description
neg_codebooks	The negative calc_codebooks

A.2.51. GETPARTS

Calculates the segments of a window

Syntax: `[ xsteps, ysteps ] = getParts( minX, minY, maxX, maxY, num_parts)`

Inputs:

Name	Description
minX	Lowest x value
minY	Lowest y value
maxX	Highest x value
maxY	Highest y value
num_parts	(Even) number of segments

Outputs:

Name	Description
ysteps	Y offsets inside the bounding box [[from; to], ...] (2xnum_parts Matrix)
xsteps	X offsets inside the bounding box [[from; to], ...] (2xnum_parts Matrix)

A.2.52. JOIN_PART_COVARIANCES

Joins multiple small files into one matrix

Syntax: `neg_model = join_part_covariances(num_parts)`

Inputs:

Name	Description
num_parts	Number of files to join

Outputs:

Name	Description
neg_model	the negative model for whitened HoGs

A.2.53. REDUCE_MATCHES

Reduces the amount of detected matches with a non-max suppression

Syntax: `[bbox, scores, idx] = reduce_matches(params, bbox, scores)`

Inputs:

Name	Description
params	Configuration parameters
bbox	Nx4 matrix of bounding boxes [x, y, w, h]
scores	N dimensional vector of scores

Outputs:

Name	Description
bbox	New bounding boxes
scores	New scores
idx	Mapping between input and output (index vector)

A.2.54. GETHOGSINSIDEBOX

Restrict calculated HoGs to a given mask

Syntax: `[X, W, M, offsets, uus, vvs, scales] = getHogsInsideBox(t, I, mask, pa`

Inputs:

Name	Description
params	Configuration parameters
I	Source image of features
t	Feature pyramid
mask	logical mask to restrict features (same size as source image)

Outputs:

Name	Description
W	wavelet
X	Features (Nx775)
M	Gradient
uus	Patch positions
vvs	Patch positions
scales	Patch scales
offsets	Patch offsets

A.2.55. WHITEN_FEATURES

Transforms HoG features into whitened HoGs

Syntax: `features = whiten_features( params, features )`

Inputs:

Name	Description
params	Configuration struct
features	Feature struct array. Required Fields: X, M

Outputs:

Name	Description
features	Whitened feature struct array. New Field: distVec

A.2.56. MERGE_STRUCTS

Merges multiple structs togetherFields of the first struct gets overriden by the second, the third, ...

Syntax: `result = merge_structs(struct1, struct2, ...)`

Outputs:

Name	Description
result	The merged struct

A.2.57. CLEAN_STRUCT

Removes all fields which shouldn't be stored in a file

Syntax: `out = clean_struct(in, remove_fields)`

Inputs:

Name	Description
in	input struct
remove_fields	Additional field list to remove

Outputs:

Name	Description
out	resulting struct

A.2.58. PROFILE_STEP

Logs an execution traceLogs where it was called and which time elapsed since the profiling start

Syntax: `params = profile_log( params )`

Inputs:

Name	Description
params	A configuration struct produced by profile_start

Outputs:

Name	Description
params	The input struct with updated profile.steps field (not required for subsequential calls)

A.2.59. MAKE_SOUND

Plays a soundtrack to get attentionCould be used to signal the end of a computation or the presence of an error

Syntax: `make_sound( finished )`

Inputs:

Name	Description
finished	Boolean, specifies if a gong (false) or a handel (true) should be played

A.2.60. SAVE_EX

Advanced wrapper around saveProvides status information and to serialize the variables with hlp_serialize

Syntax: `save_ex(filename, save_args, ...)`

Inputs:

Name	Description
	noserialize - Disable serialization (default)
save_args	Variadic arguments for matlabs save function
filename	The file to save to

A.2.61. ALPHA_BLEND

Blends 2 images together based on an alpha valueC = alpha * A + (1 - alpha) * B

Syntax: `C = alpha_blend(A, B, alpha[, mask])`

Inputs:

Name	Description
B	Image B (height, width[, colorchannels])
mask	Optional logical mask where blending should occur
A	Image A (height, width[, colorchannels])
alpha	Value between [0, 1]

A.2.62. PARSE_KEYWORDS

Parses a list of arguments into a structRequires arguments of the form Keyword1, Value1, Keyword2, Value2, ...

Syntax: `keywords = parse_keywords(input_args, allowed_keywords)`

Inputs:

Name	Description
allowed_keywords	Optional cell array of allowed keywords
input_args	Cell array of input arguments (e.g. varargin)

Outputs:

Name	Description
keywords	Struct of keyword, value pairs

A.2.63. FILE_CACHE_ENABLED

Is the file cache enabled?

Syntax: `[CACHE_FILE, params] = file_cache_enabled(params)`

Inputs:

Name	Description
params	Updated params struct if some fields are missing
CACHE_FILE	True if data should be stored to a file

Outputs:

Name	Description
params	Updated params struct if some fields are missing
CACHE_FILE	True if data should be stored to a file

A.2.64. LOAD_EX

Advanced load wrapperPrints status information and allows to load files serialized with hlp_deserializeBehaves exactly like matlabs load function

Syntax: `out = load_ex(filename, load_arg, ...)`

Inputs:

Name	Description
load_arg	Optional, variadic arguments for matlabs load function
filename	File to load

Outputs:

Name	Description
serialized	optional boolean indicating the load of a serialized var
out	optional struct containing the loaded variables

A.2.65. STRUCT2STR

converts a struct into a printable string

Syntax: `str = struct2str(in, recursive)`

Inputs:

Name	Description
in	the struct to print
recursive	boolean to indicate a recursive print

Outputs:

Name	Description
str	String containing a text representation of the struct

A.2.66. PROFILE_STOP

Stops the profilingRecords the total execution time during the profiling

Syntax: `profile_stop( params )`

Inputs:

Name	Description
params	The configuration struct processed by profile_start

A.2.67. LOAD_GROUNDTRUTH

Loads the groundtruth from the database list

Syntax: `files = load_groundtruth(params)`

Inputs:

Name	Description
params	Configuration with dataset, class and db_stream_name set

Outputs:

Name	Description
files	Struct array if files with fields: curid, I, positive, bbox, objectid

A.2.68. IMAGE_WITH_OVERLAY

Draws a 40

Syntax: `I2 = image_with_overlay(I, bbs)`

Inputs:

Name	Description
I	Image to modify
bbs	Bounding box of area which shouldn't be overlayed

Outputs:

Name	Description
I2	Overlayed image

A.2.69. PROFILE_START

Starts an execution traceupdates the given configuration struct by adding a profile field with subfieldsrecords the start time.

Syntax: `params = profile_start( params )`

Inputs:

Name	Description
params	A configuration struct with a configured dataset (dataset.localdir field is required)

Outputs:

Name	Description
params	The updated struct. Required for profile_log and profile_stop calls

A.2.70. ALLOC_STRUCT_ARRAY

Allocates a struct array of given size with given fieldsSorts fieldnames to be in line with matlabs save -struct

Syntax: `array = alloc_struct_array( size, field, ... )`

Inputs:

Name	Description
size	The requested size of the struct array (vector possible)
field	Variable number of fields to be contained in the array

Outputs:

Name	Description
array	The 1xsize struct array

A.2.71. SET_LOG_LEVEL

set current log level

Syntax: `set_log_level(lvl)`

Inputs:

Name	Description
lvl	set the current log level

A.2.72. GET_LOG_LEVEL

get current log level as string

Syntax: `lvl = get_log_level()`

Outputs:

Name	Description
lvl	get the current log level

A.2.73. DEBG

log a debug message

Syntax: `debg(fmt, ..., [addprefix], [addnewline])`

Inputs:

Name	Description
addprefix	optional boolean to indicate if the prefix should be prepended
addnewline	optional boolean to indicate if a new line should be appended
fmt	Message to log. Formatting available
updateline	optional boolean to indicate if the previous line should be overwritten

A.2.74. ERR

log a error message

Syntax: `err(fmt, ..., [addprefix], [addnewline])`

Inputs:

Name	Description
addprefix	optional boolean to indicate if the prefix should be prepended
addnewline	optional boolean to indicate if a new line should be appended
fmt	Message to log. Formatting available
updateline	optional boolean to indicate if the previous line should be overwritten

A.2.75. LOG_FILE

gets or sets the log file

Syntax: `fname = log_file([filename])`

Inputs:

Name	Description
filename	optional filename to set

Outputs:

Name	Description
fname	current logfile

A.2.76. LOG_LEVEL

gets or sets the log level

Syntax: `ll = log_level([level])`

Inputs:

Name	Description
level	optional log level to set

Outputs:

Name	Description
ll	current log level

A.2.77. WARN

log a warning message

Syntax: `warn(fmt, ..., [addprefix], [addnewline])`

Inputs:

Name	Description
addprefix	optional boolean to indicate if the prefix should be prepended
addnewline	optional boolean to indicate if a new line should be appended
fmt	Message to log. Formatting available
updateline	optional boolean to indicate if the previous line should be overwritten

A.2.78. INFO

log a info message

Syntax: `info(fmt, ..., [addprefix], [addnewline])`

Inputs:

Name	Description
addprefix	optional boolean to indicate if the prefix should be prepended
addnewline	optional boolean to indicate if a new line should be appended
fmt	Message to log. Formatting available
updateline	optional boolean to indicate if the previous line should be overwritten

A.2.79. LOG_MSG

internal logging function

Syntax: `log_msg(newfmt, fmt, ..., [updateline], [addprefix], [addnewline])`

Inputs:

Name	Description
addprefix	optional boolean to indicate if the prefix should be prepended
updateline	optional boolean to indicate if the previous line should be overwritten
fmt	Message to log. Formatting available
newfmt	Prefix format
addnewline	optional boolean to indicate if a new line should be appended

A.2.80. SUCC

log a success message

Syntax: `succ(fmt, ..., [addprefix], [addnewline])`

Inputs:

Name	Description
addprefix	optional boolean to indicate if the prefix should be prepended
addnewline	optional boolean to indicate if a new line should be appended
fmt	Message to log. Formatting available
updateline	optional boolean to indicate if the previous line should be overwritten

A.2.81. RECONSTRUCT_MATRIX_BY_OVERWRITE

reconstructs a given sparse matrix into a full integral image

Syntax: `outmatrix = reconstruct_matrix_by_overwrite(integral)`

Inputs:

Name	Description
integral	integral struct with fields scores, coords, I_size

Outputs:

Name	Description
outmatrix	Expanded integral matrix

A.2.82. SPARSE_CODEBOOK_INTEGRAL

calculates a codebook entry from a kd-Tee based integral image

Syntax: `codebook = sparse_codebook_integral(integral, queryX, queryY)`

Inputs:

Name	Description
integral	integral struct with fields tree.x and tree.y. Scores mustnot be summed up beforehand.
x, y	x and y coordinates (2 elements per vector)

Outputs:

Name	Description
codebook	Codebook vector

A.2.83. RECONSTRUCT_MATRIX

reconstructs a given sparse matrix into a full integral image

Syntax: `outmatrix = reconstruct_matrix(inmatrix)`

Inputs:

Name	Description
inmatrix	DxWxH Matrix with changed cells filled, everything else 0

Outputs:

Name	Description
outmatrix	The same matrix but expanded into a integral matrix.Input matrix is reused for speed

A.2.84. SPARSE_CODEBOOK

calculates a codebook entry from a kd-Tee based integral image

Syntax: `codebook = sparse_codebook(integral, queryX, queryY)`

Inputs:

Name	Description
integral	integral struct with fields tree.x and tree.y. Scores mustnot be summed up beforehand.
queryX, queryY	requested coordinates

Outputs:

Name	Description
codebook	Codebook vector

A.2.85. FILTER_WINDOWS_KDTREE

Filters a list of windows based on the amount of codebook dimensions set

Syntax: `filter = filter_windows_kdtree(tree, windows, dimensions, parts, codebooks)`

Inputs:

Name	Description
codebooks	Number of dimensions per codebook
parts	Number of parts per window
windows	4xN window list
dimensions	vector of relevant codebook dimensions
tree	kd-Tree to use

Outputs:

Name	Description
filter	Logical vector of remaining windows

A.2.86. RECONSTRUCT_MATRIX_BY_SUM

reconstructs a given sparse matrix into a full integral image

Syntax: `outmatrix = reconstruct_matrix_by_sum(integral)`

Inputs:

Name	Description
integral	integral struct with fields scores, coords, I_size. Scores mustnot be summed up beforehand.

Outputs:

Name	Description
outmatrix	Expanded integral matrix

A.2.87. GET_CURRENT_SCALES_BY_INDEX

get the current scales by an index

Syntax: `current_scales = get_current_scales_by_index(si, unique_scales, scale_`

Inputs:

Name	Description
scale_sizes	amount of scales per split
si	index
unique_scales	available scales

Outputs:

Name	Description
current_scales	unique list of scales

A.2.88. SORT_CACHE_FILES

sort a list of cache files by sizes extracted by the given format

Syntax: `[files, sizes] = sort_cache_files(files, format)`

Inputs:

Name	Description
format	format string to extract the sizes
files	list of possible files

Outputs:

Name	Description
sizes	list of extracted sizes
files	list of resorted files

A.2.89. FILTER_FEATURE_BY_SCALE

filters given features by a list of of given scales

Syntax: `filtered = filter_feature_by_scale(current_scales, feature)`

Inputs:

Name	Description
current_scales	Vector of possible scales
feature	feature struct

Outputs:

Name	Description
filtered	logic vector

A.2.90. GET_AVAILABLE_SCALES

get the available scales by this feature struct

Syntax: `[unique_scales, scale_sizes] = get_available_scales(params, feature)`

Inputs:

Name	Description
params	Configuration struct
feature	feature struct

Outputs:

Name	Description
scale_sizes	amount of scales per split
unique_scales	unique list of scales

A.2.91. FILTER_CACHE_FILES

filters a list of cache files based on the best matching query size

Syntax: `filename = filter_cache_files(params, files, sizes, requested_size)`

Inputs:

Name	Description
params	Configuration struct
requested_size	size of query part
sizes	corresponding sizes
files	list of possible files

Outputs:

Name	Description
filename	path of best matching file

A.2.92. GET_CURRENT_SCALES_BY_SIZE

get the current scales by a query size

Syntax: `current_scales = get_current_scales_by_size(params, unique_scales, sca`

Inputs:

Name	Description
params	Configuration struct
scale_sizes	amount of scales per split
roi_sizes	query size
unique_scales	available scales

Outputs:

Name	Description
current_scales	unique list of scales

A.2.93. GET_POSSIBLE_CACHE_FILES

get all possible cache files based on a format string

Syntax: `files = get_possible_cache_files(cachename)`

Inputs:

Name	Description
cachename	format string to search for. Replaces

Outputs:

Name	**Description**
files	list of possible files