MGR-Project-Code Documentation

Release 1.0

Author: Vit Ruzicka

Jul 21, 2017

Contents:

CHAPTER 1

DatasetHandler package

1.1 Submodules

1.2 DatasetHandler.CreateDataset module

DatasetHandler.CreateDataset.**determineUniqueId**(*dataset_nickname*, *desired_number*, *seed*)

Each dataset needs its own unique name build from the parameters which influence its shape. Later we name the feature files with this name and thus it needs to be linked with each dataset. :param dataset_nickname: id :param desired_number: number of images, can be None if all :param seed: seed for random data shuffling :return: string of unique id

DatasetHandler.CreateDataset.**get_path_for_dataset**(*folder*, *filename_override*)

Get path to dataset from couple of default values, or use suggested name :param folder: name of the folder, will be joined to the project path :param filename_override: suggested name of dataset dump file :return: path

DatasetHandler.CreateDataset.**load_custom**(*folder*, *pixels*, *desired_number=None*, *seed=None*, *filename_override=''*)

Load dataset from one of prepared folders. Main method to load datasets. :param folder: folder name inside StreetViewData :param pixels: pixel size :param desired_number: number of images, can be None if all :param seed: seed for random data shuffling :param filename_override: if the name is not SegmentsData_marked_R100.dump or SegmentsData.dump :return: dataset object

DatasetHandler.CreateDataset.**prepareDataset**(*path*, *dims*, *desired_number*, *seed*)

Create dataset object and prepare it from the suggested Segments file. :param path: path :param dims: pixel sizes list (width and height) :param desired_number: number of images, can be None if all :param seed: seed for random data shuffling :return: dataset object

1.3 DatasetHandler.DataAugmentation module

DatasetHandler.DataAugmentation.**handle_noncanon_dataset**(*Settings*, *model_settings*)

Special case scenario. We are creating a new custom dataset, instead of using one of the big officially used,

"canon" datasets :param Settings: Setting for the whole experiment :param model_settings: Setting for our one dataset :return:

1.4 DatasetHandler.DatasetObj module

class DatasetHandler.DatasetObj.**Dataset**

Common base class for a Dataset

What does dataset have?

- source data folder with images and their scores - via Segments, but we don't use Segments anymore anywhere out

What can it do?

- give it's data nicely out ([x],[y]) as image data and labels
- give us subsets, which are uniform in sense of Scoring
- provide us with statistics - without worrying about unsuccessful downloads or anything
- give us *views* like exporting images into folder for inspection - for example coded with score

DumpFilesIntoDirectory_withScores(*target_directory=''*)

Simple way of visualizing which images are considered "attractive" (with high score) and which are not :param target_directory: target directory, for example target_directory = '../debugViewOfDataset/' :return: returns list of new names of files, with the order unchanged

MapScoreToImages(*into_bins=100*)

Gets a dict which give to a index from 0-100 a list of images of such score (score goes in range 0-1, so *100 in this case) :return:

cast_osm_to_bool()

Transforms the osm vector data to boolean values, aka i=0 -> 0, i>0 -> 1 :return:

cast_osm_to_one_hot_categories()

Transforms the osm vector data to one hot categories - low,mid,high represented as 001,010,100 binary. :return:

debug_print_first(*n*)

Debug print first n values in this dataset.

expandOsmDataWithMultipleRadii(*model_settings*)

idea is to load all the radii data we have available and add it to each of the segments # we assume the basic experiment definition

flag_is_extended = False

generator_images_scores(*order*, *image_paths*, *scores*, *resize=None*)

Get generator of images :param order: prearanged order (1,2,3...) or (2,55,1,980, ...) :param image_paths: paths to images, these are kept in memory while the big 640x640x3 image data is not :param scores: score to be associated with returned image :param resize: parameter to resize loaded images on the fly :return: generator, which yields (image, score)

getDataLabels(*resize=None*)

([x,y]) as image data and labels

getDataLabels_only_osm()

Get just osm data

getDataLabels_only_osm_raw()
return list of osm without coversion to numpy format

getDataLabels_only_y()
Get just label data

getDataLabels_split(*resize=None*, *validation_split=0.2*)
([x, y, x_val, y_val]) as image data and labels after being split

getDataLabels_split_only_osm(*validation_split=0.2*)
Get just osm data, after validation split

getDataLabels_split_only_y(*resize=None*, *validation_split=0.2*)
Get just label data, after validation split

getImageGenerator(*validation_split*, *resize=None*)
Return generators # take the lists on images and their labels - split these two arrays by the validation split

getShapeOfOsm()
Get shape of the osm data - aka dimension of the vectors, traditionally (594,) needed for building Keras models. :return:

has_osm_loaded = False

img_height = -1

img_width = -1

init_from_lists(*list_of_images*, *labels*, *osm*, *segment_ids*, *img_width*, *img_height*)
Initialization from lists of data

init_from_segments(*path_to_segments_file*, *img_width*, *img_height*)
Initialization from loaded Segment in path_to_segments_file # Segments are not used apart from initialization

log_the_osm()
Apply log to values in OSM vector. :return:

num_of_images = 0

plotHistogram(*save_to_pdf=False*, *labels_override=None*)
Plot score of this dataset as histogram. :param save_to_pdf: flag to save into output.pdf :return:

randomize_all_list_order_deterministically_same_segment(*local_seed*)

According to a chosen seed number will shuffle contents of lists (of urls, of scores, of osm) so they are kept intact.

Returns

remove_dual_osms()
We know that every third entry is actually unique (in sense of osm,score pair) - when we are talking about osm_only model :return:

sampleUniform(*desired_number*)
randomized subsample of a dataset

spawnUniformSubset(*desired_number*)
Spawn a subset from dataset uniform distribution over the original data.

Parameters **desired_number** – size of the desired dataset

Returns the resulting new dataset

statistics()
Report important information about the dataset.

test_existence_of_all_images()
Test physical presence of the images - useful for debuging. :return:

unique_id = ''

1.5 DatasetHandler.DatasetVizualizators module

DatasetHandler.DatasetVizualizators.**GenerateAverageImagesFromDictionary**(*dict*, *save_to_dir=None*, *output_folder=None*)
Gets a dictionary of d[score_label_value] pointing to an array of images :param dict: :return: Up to 100 averaged images

DatasetHandler.DatasetVizualizators.**plotHistogram**(*x*, *title=''*, *num_bins=100*, *x_min=0.0*, *x_max=1.0*, *custom_x_label='Score value'*, *custom_y_label='Count of occurances'*)
Plot histogram from the x data.

DatasetHandler.DatasetVizualizators.**plotMultipleWhiskerPlots**(*datas*, *whiskers*, *labels*)
Example run:

means_men = (20, 35, 30, 35, 27) std_men = (2, 3, 4, 1, 2) means_women = (25, 32, 34, 20, 25) std_women = (3, 5, 2, 3, 3)

datas = [means_men, means_women, means_men] whiskers = [std_men, std_women, std_women] labels = ['1', '2', '3']

plotMultipleWhiskerPlots(datas,whiskers,labels)

DatasetHandler.DatasetVizualizators.**plotWhisker**(*data*, *title=''*, *y_min=0.0*, *y_max=1.0*, *legend_on=True*, *notch=True*)
Plot box plot / whisker graph from data.

DatasetHandler.DatasetVizualizators.**plotX_sortValues**(*dont_touch_this_x*, *title=''*, *x_min=0.0*, *x_max=1.0*, *notReverse=False*, *custom_x_label='# of images'*, *custom_y_label='Score value'*)
Visualization of dataset by the method of sorting array by value and plotting.

DatasetHandler.DatasetVizualizators.**saveAllPlotsToPDF**()
Save all created plots into a pdf file.

DatasetHandler.DatasetVizualizators.**show**()
show plots on screen

DatasetHandler.DatasetVizualizators.**subPlot2**(*fce1*, *fce2*, *param1=None*, *param2=None*)
Join two plots.

Example run: def tmp_fce1(): ... def tmp_fce2(): ... subPlot2(tmp_fce1, tmp_fce2)

DatasetHandler.DatasetVizualizators.**xkcd**()
special style

DatasetHandler.DatasetVizualizators.**zoomOut**(*axes*, *xlim=None*, *ylim=None*, *factor=0.05*)
Set size to fit in limitations. :param axes: handler to matlibplot :param xlim: list of [from x, to x] values :param ylim: list of [from y, to y] values :param factor: zoom factor :return:

DatasetHandler.DatasetVizualizators.**zoomOutX**(*axes*, *xlim=None*, *factor=0.05*)
handle the X axis

DatasetHandler.DatasetVizualizators.**zoomOutY**(*axes*, *ylim=None*, *factor=0.05*, *only_up=False*)
handle the Y axis

1.6 DatasetHandler.FileHelperFunc module

DatasetHandler.FileHelperFunc.**copy_file**(*src*, *dst*)
Copy and paste file.

DatasetHandler.FileHelperFunc.**copy_folder**(*src*, *dst*)
Copy and paste folders. Used for dataset augmentation.

DatasetHandler.FileHelperFunc.**file_exists**(*fname*)
Does file exist, returns boolean.

DatasetHandler.FileHelperFunc.**folder_exists**(*directory*)
Does folder with this name exist, returns boolean

DatasetHandler.FileHelperFunc.**get_folder_from_file**(*fname*)
Get folder name from path to a file.

DatasetHandler.FileHelperFunc.**get_geojson_path**()
Gives us the path directly to attractivity_previtus_data_1_edges.geojson from a list of allowed paths :return:

DatasetHandler.FileHelperFunc.**get_project_folder**()
Gives us the path to MGR-Project-Code from a list of allowed folders. :return:

DatasetHandler.FileHelperFunc.**make_folder_ifItDoesntExist**(*directory*)
Make a new directory, if it didn't previously exist.

DatasetHandler.FileHelperFunc.**md5**(*fname*)
Get md5 hash of a file.

DatasetHandler.FileHelperFunc.**use_path_which_exists**(*list_of_possible_paths*)
From a list of possible paths choose the one which exists. :param list_of_possible_paths: possible paths :return: working path

1.7 Module contents

CHAPTER 2

Downloader package

2.1 Subpackages

2.1.1 Downloader.PreprocessData package

2.1.1.1 Submodules

2.1.1.2 Downloader.PreprocessData.DecoratorRetry module

Downloader.PreprocessData.DecoratorRetry.**retry**(*ExceptionToCheck*, *tries=4*, *delay=3*, *backoff=2*, *logger=None*)

Retry calling the decorated function using an exponential backoff.

http://www.saltycrane.com/blog/2009/11/trying-out-retry-decorator-python/ original from: http://wiki.python.org/moin/PythonDecoratorLibrary#Retry

Parameters

- **ExceptionToCheck** (*Exception or tuple*) – the exception to check. may be a tuple of exceptions to check
- **tries** (*int*) – number of times to try (not retry) before giving up
- **delay** (*int*) – initial delay between retries in seconds
- **backoff** (*int*) – backoff multiplier e.g. value of 2 will double the delay each retry
- **logger** (*logging.Logger instance*) – logger to use. If None, print

2.1.1.3 Downloader.PreprocessData.DownloadUrlFilenameMap module

Downloader.PreprocessData.DownloadUrlFilenameMap.**DownloadUrlFilenameMap**(*FilenameMap*, *Segments*)

Download multiple files according to the FilenameMap. List of tripples in [(<url>, <filename>, <edge id>), ...

]

Marks entires in Segments with flags about the images. Returns list of failed downloads

Downloader.PreprocessData.DownloadUrlFilenameMap.**md5**(*fname*)

Downloader.PreprocessData.DownloadUrlFilenameMap.**urlretrieve_with_retry**(**args*, ***kwargs*)

Saves (image) file and returns (<filename_string>, <object of httplib.HTTPMessage>)

2.1.1.4 Downloader.PreprocessData.Functions module

Downloader.PreprocessData.Functions.**bearing_between_two_points**(*start*, *end*, *degrees_offset=0.0*)

Calculates the initial bearing between two geographical locations. see: http://www.movable-type.co.uk/scripts/latlong.html The bearing is angular distance from NORTH.

Downloader.PreprocessData.Functions.**distance_between_two_points**(*start*, *end*)

Calculate distance between start and end :param start: :param end: :return: the distance

Downloader.PreprocessData.Functions.**getApi**()

Downloader.PreprocessData.Functions.**interpolation**(*start*, *end*, *fraction*)

Interpolate a custom fraction between points. :param start: :param end: :param fraction: from 0 to 1 :return:

Downloader.PreprocessData.Functions.**midpoint**(*start*, *end*)

Interpolate a midpoint between two points. :param start: :param end: :return: lat and lot of new point

Downloader.PreprocessData.Functions.**segmentIDtoListID**(*semgentId*)

segment id might be 1000 if we start there, but the list is indexing from 0

2.1.1.5 Downloader.PreprocessData.GenListOfUrls module

Downloader.PreprocessData.GenListOfUrls.**GenListOfUrls**(*Segments*, *PIXELS_X*, *PIXELS_Y*, *PrependPath=''*, *minimal_length=20*, *custom=False*)

Iterates over the segment list and returns a list of urls needed for download Outputs list of tripples in [(<url>, <filename>, <edge id>), ...]

2.1.1.6 Downloader.PreprocessData.PrepSegments module

Downloader.PreprocessData.PrepSegments.**PrepSegments**(*EdgesGEOJSON*, *FromEdgeID=0*, *ToEdgeID=2*)

Alternative loading method, which relies just upon edge data (we don't have nodes fully available this time).

Parameters **EdgesGEOJSON** – Inputs the json object obtained via json.load(json_file)

Returns Returns the whole set of Segments

2.1.1.7 Downloader.PreprocessData.SegmentObj module

class Downloader.PreprocessData.SegmentObj.**SegmentObj**(*Start*, *End*, *Score*, *SegmentId*)

Common base class for segment Variables: FromId, ToId, SegmentId Functions: getBearingString(), getGoogleViewUrl()

ScoreAdjustment(*val*)

betweenPoints(*pointStart*, *pointEnd*)
Generate urls and filenames in between these two points. They will be added to the rest of data in this Segment. :param pointStart: :param pointEnd: :return: urls and filenames

checkOSMVersion(*verbose=True*)

displaySegment()

displaySegmentShort()

getBearingString(*Location*, *Direction*, *degrees_offset=0.0*)

getGoogleViewUrl(*Location*, *Direction*, *resx*, *resy*, *degrees_offset=0.0*)
Google View url from the start of this segment

getGoogleViewUrls(*PIXELS_X*, *PIXELS_Y*)
Generate urls for the Segment. Here we also reset the :param resx: Resolution of images, X :param resy: Resolution of images, Y :return: returns list of urls and filenames to download

getGoogleViewUrls_whileUsingFractionsOfMinEdgeLen(*PIXELS_X*, *PIXELS_Y*, *minimal_length*)
More sophisticated method, splitting long edges into minimal_length :param PIXELS_X: :param PIXELS_Y: :param minimal_length: minimal allowed edge len :return:

getImageFilename(*i_th_image*)
Unified filename generation

getLocation(*i_th_image*)

getNearbyVector(*i_th_image*)

getScore()

hasLoadedImageI(*i*)

hasUnknownScore()

markWithVector(*nearby_vector*, *index*, *MARKING_VERSION*)

resetImageMemory(*number_of_images*)

setScore(*s*)

2.1.1.8 Downloader.PreprocessData.SegmentsManipulators module

Downloader.PreprocessData.SegmentsManipulators.**AdditionalStatistics**(*Segments*)

Downloader.PreprocessData.SegmentsManipulators.**StatisticsSegments**(*Segments*, *additionalStatistics=False*)
Provide statistics for loaded dataset :param Segments: input list of Segments

2.1.1.9 Module contents

2.2 Submodules

2.3 Downloader.DataOperations module

Downloader.DataOperations.**FixDataFile_FailedDownloads**(*name*, *ERROR_TYPE*, *PIXELS_X*, *PIXELS_Y*, *PrependPath*)

Loads, fixes and saves the structure of Segments. Looks at those with particular error messages ERROR_TYPE and redownloads images. Returns fixed Segments while it also saves them.

Error codes in Default.py:

- ERROR_MESSAGE_NOT_FOUND = 404
- ERROR_MESSAGE_FAILED_MANY_TIMES = 101

Eample call: FixDataFile_FailedDownloads(DATASTRUCTUREFILE, ERROR_MESSAGE_FAILED_MANY_TIMES)

Downloader.DataOperations.**HasSomeErrorneousData**(*Segments*, *ERROR_TYPE*)

Check for errors. Example: if (HasSomeErrorneousData(Segments,E)):

Segments = FixDataFile_FailedDownloads(_,E)

Downloader.DataOperations.**LoadDataFile**(*name*)

Load Segments from the file <name>

Downloader.DataOperations.**MarkBadSegments**(*Segments*, *MD5_list*, *MARKERR*)

Used to manually mark bad segments after they are downloaded (using the rest of the set and not having to redownload it all). Ps: all of the known errors we mark directly while downloading in DownloadUrlFilenameMap

Example call: # manual marking and saving #Segments = MarkSegmentsWithImagesOfMD5(Segments, QUOTA_EXCEEDED_CHECKSUM, ERROR_MESSAGE_QUOTA) #SaveDataFile(DATASTRUCTUREFILE, Segments)

Parameters

- **Segments** – Input segments (remember to save them after! and possibly back them up before)
- **MD5_list** – we are looking for certain md5 of the image - for example "b2328ec7ff935944a85723daddf0e8b7" was quota
- **MARKERR** – we want to mark the *bad* segments - we will thus force all the photos related to one Segment to redownload

Returns Edited Segments, remember to save them.

Downloader.DataOperations.**SaveDataFile**(*name*, *Segments*)

Save structure of Segments into file <name>

Downloader.DataOperations.**save_segments_file_as_without_missing_files**(*in_segments_file*, *path_to_images*, *out_segments_file*)

Downloader.DataOperations.**save_segments_file_marking_missing_files_as_errors**(*in_segments_file*, *path_to_images*, *out_segments_file*)

2.4 Downloader.Defaults module

2.5 Downloader.DownloaderRunner module

Downloader.DownloaderRunner.**RunCheck**(*name*, *px_py*)
Check downloaded Segment files while downloading missing data. :param name: name of the Segments file :param px_py: pixel sizes :return:

Downloader.DownloaderRunner.**RunDownload**(*name*, *from_edge*, *to_edge*, *px_py*, *minimal_length*, *custom_geojson=''*)
Run downloader while setting the most important variables here :param name: name of the folder it will safe in Data/StreetViewData/<name> :param from_edge: start with edge id :param to_edge: end with edge id :param px_py: pixels x and y - both same :return:

Downloader.DownloaderRunner.**RunMarkBad**(*name*)
Mark errorneous segments by Error flag. :param name: :return:

2.6 Downloader.GenerateGIFAnimation module

Downloader.GenerateGIFAnimation.**GenerateGIFAnimation**(*Segments*, *GIFFileName*)
Create GIF animation from Segments.

2.7 Downloader.ImageHelpers module

Downloader.ImageHelpers.**len_**(*L*)

Downloader.ImageHelpers.**list_images**(*folder*)
prepare list of image names

Downloader.ImageHelpers.**load_image_with_keras**(*img_path*, *target_size=None*, *dim_ordering='channels_last'*)

Downloader.ImageHelpers.**load_images_with_keras**(*img_paths*, *target_size=None*, *dim_ordering='channels_last'*)

Downloader.ImageHelpers.**preprocess_image_batch**(*image_paths*, *img_size=None*, *crop_size=None*, *color_mode='rgb'*, *out=None*)

Downloader.ImageHelpers.**saveArrayToCSV**(*array*, *filename*)

2.8 Downloader.KerasPreparation module

Downloader.KerasPreparation.**LoadActualImages**(*list_of_images*, *resize=None*, *dim_ordering='channels_last'*)

Downloader.KerasPreparation.**LoadDataFromSegments**(*Segments*, *has_score=True*, *path_to_images=None*, *we_dont_care_about_missing_images=False*)
Turns loaded segments into data we will need for keras. :param Segments: Loaded segments :param path_to_images: additional path specification which we need before 'images/—.jpg' :return: Returns list of urls of images and their labels (score in the Segment)

Downloader.KerasPreparation.**split_data**(*x*, *y*, *validation_split=0.2*)

Parameters

- **x** – Dataset, can be paths to images or directly the image data for example (?, 3,222,222)
- **y** – Labels of the datasets
- **validation_split** – Split ratio, defaults to 80% for test set and 20% of validation set

Returns Returns split data

Downloader.KerasPreparation.**split_one_array**(*arr*, *validation_split=0.2*)

Parameters

- **arr** – list of data to be split
- **validation_split** – Split ratio, defaults to 80% for test set and 20% of validation set

Returns Returns split data

Downloader.KerasPreparation.**split_osm**(*osm*, *validation_split=0.2*)

Split array of osm vectors by validation split. :param osm: osm data :param validation_split: 0 to 1 fraction :return: splitted osm data into osm_test, osm_val

2.9 Downloader.PreprocessDataF module

2.10 Downloader.Tester module

2.11 Downloader.UseData module

2.12 Downloader.VisualizeHistory module

Downloader.VisualizeHistory.**loadHistory**(*filename*)

Downloader.VisualizeHistory.**saveHistory**(*history_dict*, *filename*)

Downloader.VisualizeHistory.**visualize_histories**(*histories*, *names*, *plotvalues='loss'*, *show=True*, *save=False*, *save_path=''*, *custom_title=None*, *just_val=False*)

Visualize multiple histories.

Example usage: h1 = loadHistory('history1.npy') h2 = loadHistory('history2.npy') visualize_histories([h1, h2], ['history1', 'history2'])

Downloader.VisualizeHistory.**visualize_history**(*hi*, *show=True*, *save=False*, *save_path=''*, *show_also=''*, *custom_title=None*)

Example calls: hi = model.fit(...)

saveHistory(hi.history, 'tmp_saved_history.npy') visualize_history(loadHistory('tmp_saved_history.npy'))

Downloader.VisualizeHistory.**visualize_special_histories**(*histories*, *plotvalues='loss'*, *show=True*, *save=False*, *save_path=''*, *custom_title=None*, *just_val=False*)

We are visualizing results of a k-fold crossvalidation training. In <histories> we have the individual runs of the experiment.

Downloader.VisualizeHistory.**visualize_whiskered_boxed**(*whiskered_boxes_data*, *names*, *show=True*, *save=False*, *save_path=''*, *custom_title=''*)

We are visualizing results of a k-fold crossvalidation training. In <whiskered_boxes_data> we have data for whiskered box plots.

2.13 Module contents

CHAPTER 3

Evaluator package

3.1 Submodules

3.2 Evaluator.Evaluator module

Evaluator.Evaluator.**evaluator**(*model_file*, *settings_file*, *name_output_file*, *custom_target_geojson=None*, *show_segments_histo_stats=False*, *actually_save=True*)

Main Evaluator function. :param model_file: path to model .h5 file. :param settings_file: path to settings file which was used to train this model. :param name_output_file: name of labeled geojson data :return:

Evaluator.Evaluator.**evaluator_generators_predict**(*model_base*, *model_top*, *model_settings*, *img_generator*, *osm*, *size*)

Use generators to evaluate model. :param model_base: base CNN model :param model_top: attached custom top classifier model :param model_settings: settings of the model (to recognize the type of the model) :param img_generator: image generator :param osm: list of corresponding osm vectors :param size: amount of images. :return: Returns labeled data (labeled from images and osm)

Evaluator.Evaluator.**evaluator_load_model**(*model_file*, *settings_file*, *verbose=False*)

Load skeleton of model and dataset :param model_file: model h5 file :param settings_file: corresponding settings file :param verbose: :return: model_base, model_top, model_settings

Evaluator.Evaluator.**evaluator_predict_on_dataset**(*model_base*, *model_top*, *model_settings*, *x*, *osm*)

Evaluate model by just loading data into memory. This is effective with OSM models, but with others it will likely flood the memory and be terminated - use generators in that case.

Parameters

- **model_base** – base CNN model
- **model_top** – attached custom top classifier model
- **model_settings** – settings of the model (to recognize the type of the model)

- **x** – image data
- **osm** – list of corresponding osm vectors

Returns Returns labeled data (labeled from images and osm)

Evaluator.Evaluator.**load_tmp_dataset**()
Example of how a dataset can be loaded. :return:

3.3 Evaluator.Functions module

Evaluator.Functions.**AlterSegments**(*EvaluatedData*, *Segments*, *only_unknown_scores=True*)
Edit internal values of Segments depending on what data we got. :param EvaluatedData: processed dictionary which can give us list of values for segment id :param Segments: list of Segment objects, which we iterate through and change their scores. :param only_unknown_scores: Flag whether we overwrite only those Segments which had unknown score in the initial dataset. :return: Altered Segments list

Evaluator.Functions.**analyze_lists**(*lists*)
Analyze statistics inside lists. Count for unique segments and numbers of images. :param lists: :return:

Evaluator.Functions.**default_segments_path**()
assembles path to the segments files

Evaluator.Functions.**generator_img**(*order*, *image_paths*, *resize=None*)
generator yields loaded images one by one, needed to save memory

Evaluator.Functions.**generator_osm**(*order*, *osms*)
generator yields osm vectors one by one, not really needed

Evaluator.Functions.**getImgGenerator_from_lists**(*lists*)
Create generator on given list, yielding imagery data. :param lists: :return:

Evaluator.Functions.**getOsmGenerator_from_lists**(*lists*)
Create generator on given list, yielding vector data. :param lists: :return:

Evaluator.Functions.**internalToExternal**(*score*)
Convert score notations back to how geojson file used it.

Evaluator.Functions.**loadDataFromSegments**(*path_to_segments_file*, *SCORE*, *verbose=False*, *we_dont_care_about_missing_images=False*)
Load lists from Segments :param path_to_segments_file: Segments file to be loaded. :param SCORE: flag for if we care for only scored Segments :param verbose: :param we_dont_care_about_missing_images: flag for if we care for only those Segments with images (OSM model doesnt need them.) :return: lists and Segments

Evaluator.Functions.**loadDefaultGEOJSON**()
Load default GeoJSON file, which is the initial attractivity_previtus_data_1_edges.geojson file

Evaluator.Functions.**loadGeoJson**(*path*)
Load GeoJSON file

Evaluator.Functions.**markGeoJSON**(*GeoJSON*, *Segments*)
Mark geojson object with data from corresponding Segment object in Segments :param GeoJSON: geojson object :param Segments: list of objects :return: altered GeoJSON object

Evaluator.Functions.**osm_from_lists**(*lists*)
Get osm data

Evaluator.Functions.**prepEvaluatedData**(*y_pred*, *segment_ids*)
prepare dictionary which will give us scores of certain segment id all clustered together into one list.

Evaluator.Functions.**saveGeoJson**(*GeoJSON*, *path*)
Save GeoJSON file

Evaluator.Functions.**small_lists**(*lists*, *n=50*)
Subset of first n values. :param lists: :param n: :return: subsets of lists

Evaluator.Functions.**traverseGeoJSON**(*GeoJSON*, *Segments*)
For testing purposes we go through all entries in GeoJSON and check for scores in Segments, we report the altered values. :param GeoJSON: :param Segments: :return:

3.4 Module contents

CHAPTER 4

ExperimentRunner package

4.1 Submodules

4.2 ExperimentRunner.ModelExperiments module

This is the main Experiment runner with Models

ExperimentRunner.ModelExperiments.**experiment_runner**(*settings_file=None*, *job_id=''*)
Main experiment runner function, controls the run of the whole testing scheme. :param settings_file: specification of path to Settings file :param job_id: unque id, given by the scheduling program :return:

4.3 ExperimentRunner.SettingsDefaults module

ExperimentRunner.SettingsDefaults.**load_default_settings**()
Default setting definition. :return: Default setting for whole experiment and for first model in dictionaries.

ExperimentRunner.SettingsDefaults.**load_settings_from_file**(*file=None*, *job_id=''*, *verbose=False*)
Load Settings from a custom settings description file. :param file: Load from file :param job_id: Unique id :param verbose: Flag to debug info about loaded Settings :return: Settings dictionary

ExperimentRunner.SettingsDefaults.**print_settings**(*Settings*, *ignore_default_values=True*)
Debug of Settings, prints values which are not like the ones in Default setting file. :param Settings: Custom setting to be checked :param ignore_default_values: Flag to ignore default values (for better clarity) :return:

4.4 Module contents

CHAPTER 5

ModelHandler package

5.1 Subpackages

5.1.1 ModelHandler.CreateModel package

5.1.1.1 Subpackages

5.1.1.2 Submodules

5.1.1.3 ModelHandler.CreateModel.KerasApplicationsModels module

ModelHandler.CreateModel.KerasApplicationsModels.**get_model**(*name*, *pixels=None*)

ModelHandler.CreateModel.KerasApplicationsModels.**inception_v3**(*input_shape=None*)

ModelHandler.CreateModel.KerasApplicationsModels.**resnet50**(*input_shape=None*)

ModelHandler.CreateModel.KerasApplicationsModels.**vgg16**(*input_shape=None*)

ModelHandler.CreateModel.KerasApplicationsModels.**vgg19**(*input_shape=None*)

ModelHandler.CreateModel.KerasApplicationsModels.**xception**(*input_shape=None*)

5.1.1.4 Module contents

5.2 Submodules

5.3 ModelHandler.KfoldTester module

ModelHandler.KfoldTester.**best_min**(*arr*)
return the smallest value

ModelHandler.KfoldTester.**chunks**(*l*, *n*)
Chunk data from list l into n fjords.

ModelHandler.KfoldTester.**indices_to_data**(*any_indices*, *any_data*)

ModelHandler.KfoldTester.**k_fold_crossvalidation**(*model*, *dataset*, *model_settings*)
K fold crossvalidation scheme # includes proper loading of models, testing and processing of the results.

ModelHandler.KfoldTester.**kfold**(*indices_in_fjords*, *selected*)
indices come like [] [] [] ... [], we want to select the one in <selected> as validation and rest as tests

ModelHandler.KfoldTester.**select_data**(*indices*, *data*)
select from data, while considering that data can be either list of items to directly select from # or it can be a list of size two, where we select from both items and later join

5.4 ModelHandler.MergeModelOsmImg module

5.5 ModelHandler.ModelGenerator module

ModelHandler.ModelGenerator.**build_finetune_model**(*cnn*, *top*, *cut*, *input_shape*)

ModelHandler.ModelGenerator.**build_full_mixed_model**(*osm_shape*, *img_shape=<tf.Tensor 'input_1:0' shape=(?, 299, 299, 3) dtype=float32>*)
Tests with the mixed model without using features for cooking. :param osm_shape: shape of the osm data :param img_shape: shape of the imgs data :return:

ModelHandler.ModelGenerator.**build_img_only_top_model**(*input_shape*, *number_of_repeats*)
Builds simple model of repeated FC blocks. :param input_shape: Keras needs to know the input shape. :param number_of_repeats: repeats of FC block additional to the first Flatten layer and the last sigmoid output layer. :return:

ModelHandler.ModelGenerator.**build_img_osm_mix_model**(*input_shape_img*, *input_shape_osm*, *number_of_repeats*)
Build a combined model using both imgs and osm data. :param input_shape_img: Keras needs to know the input shapes for imgs. :param input_shape_osm: Keras needs to know the input shapes for osm vectors. :param number_of_repeats: repeats of FC block additional to the first Flatten layer and the last sigmoid output layer. :return:

ModelHandler.ModelGenerator.**build_img_osm_mix_model_custom_base_cnn_top**(*input_shape_img*, *input_shape_osm*, *number_of_repeats*)

ModelHandler.ModelGenerator.**build_osm_only_model**(*input_shape*, *number_of_repeats*, *manual_width=256*)
Build a simple model with just OSM vector as it's input, couple of FC blocks and then sigmoid output of 1 score value :param input_shape: Keras needs to know the input shape. :param number_of_repeats: repeats of FC block additional to the first Flatten layer and the last sigmoid output layer. :param manual_width: manual width of the fc layers :return:

ModelHandler.ModelGenerator.**get_cnn_models**(*Settings*)
Loads the base CNN part of models :param Settings: settings used to get individual model settings :return:

ModelHandler.ModelGenerator.**get_top_models**(*models*, *datasets*, *Settings*)
Adds the right top models, now with proper knowledge of shapes of feature files. :param models: list of models where we will add new ones :param datasets: using information about databases :param Settings: and using information from Settings :return: models are generated and returned

ModelHandler.ModelGenerator.**join_two_models**(*one*, *two*)
Join two models in a chain after each other :param one: first model :param two: second model :return: joined model

ModelHandler.ModelGenerator.**report_models**(*models*, *Settings*)
Debug method - report all models :param models: list of models :param Settings: settings files to get the individual model settings :return:

ModelHandler.ModelGenerator.**split_model**(*model*, *start*, *end*)
Split model by the start and end flags - we get the subset of a model. # Thanks to https://github.com/fchollet/keras/issues/5074#issuecomment-274259404

Parameters

- **model** – model to be split
- **start** – from layer
- **end** – to layer

Returns subset of the model

5.6 ModelHandler.ModelOI module

ModelHandler.ModelOI.**do_we_need_to_cook**(*filename_features_train*, *filename_features_test*)
Checks for the existence of cooked feature files. :param filename_features_train: path to training features :param filename_features_test: path to testing features :return:

ModelHandler.ModelOI.**generate_report_string**(*Settings*)
Generation of text based report. :param Settings: main settings used for the experiment. :return:

ModelHandler.ModelOI.**getLogDirectory**()
Get established Log directories. Code will use the one of these paths which is available on the machine. :return: working path to Logs directory

ModelHandler.ModelOI.**getSharedDirectory**()
Get established Log directories. Code will use the one of these paths which is available on the machine. :return: working path to Logs/shared directory

ModelHandler.ModelOI.**get_feature_file_names**(*shared_folder*, *dataset_uid*, *model_name*, *cut=0*)

Parameters

- **shared_folder** – taken from getSharedDirectory()
- **dataset_uid** – taken from dataset.unique_id
- **model_name** – can be for example 'resnet50'
- **cut** – special case scenario, if we were cutting base CNN shorter

Returns

ModelHandler.ModelOI.**graph_histories**(*histories*, *Settings*)
Graphs histories according to Settings["graph_histories"] ~ ['all',[]] #['all',[],[0,2]] :param histories: histories to be graphed :param Settings: Main setting dict containing the path values :return:

ModelHandler.ModelOI.**load_dataset**(*Settings*)
Loads datasets according to the Settings parameters "dataset_name", "pixels", "number_of_images", "seed". Also manages shuffling and other initial editations of the dataset. :param Settings: :return: dataset object

ModelHandler.ModelOI.**load_model**(*path*)
Load Keras model from path. :param path: :return: the loaded model

ModelHandler.ModelOI.**prepare_folders**(*Settings*, *datasets*, *verbose=False*)
Figures folder paths which will be used in experiment (local folder with history file, graphs, model file and also the shared folder where features are saved. Also the paths for filename_features_train, filename_features_test for for each model. Saves these paths into Settings. :param Settings: This is a travelling dictionary with all the settings, we will add folder settings there :return:

ModelHandler.ModelOI.**save_histories**(*histories*, *Settings*)
Save histories into .npy files, which can be used to reproduce the results. :param histories: histories to be saved :param Settings: Main setting dict containing the path values :return:

ModelHandler.ModelOI.**save_metacentrum_report**(*Settings*)
Downloads and saves the Metacentrum generated file, according to the unique id value in Settings. :param Settings: :return:

ModelHandler.ModelOI.**save_models**(*models*, *Settings*)
Saves the trained models alongside with the experiments settings. :param models: list of models, each can contain base cnn model or just the top model. We differentiate according according to the Settings of model_type. :param Settings: :return:

ModelHandler.ModelOI.**save_report**(*Settings*)
Saves report of the most important settings. :param Settings: :return:

ModelHandler.ModelOI.**save_visualizations**(*models*, *Settings*)
Save visualizations of the models, if we have set it in Settings :param models: list of models to be plotted :param Settings: the main Setting, used to access output paths :return:

ModelHandler.ModelOI.**send_mail_with_graph**(*Settings*)
Reporting method by sending mail with graph as an attachment. :param Settings: :return:

5.7 ModelHandler.ModelTester module

class ModelHandler.ModelTester.**RunMonitor**(***opt*)
Bases: keras.callbacks.Callback

CUSTOM KERAS CALLBACK # inspired by Kerutils at https://github.com/samyzaf/kerutils

on_epoch_begin(*epoch*, *logs={}*)

on_epoch_end(*epoch*, *logs={}*)

on_train_begin(*logs={}*)

on_train_end(*logs={}*)

print_params()

ModelHandler.ModelTester.**cook_features**(*models*, *datasets*, *Settings*)
Makes sure that we have features available for the duo of model-dataset in our shared feature folder. If not, we will cook them. :param models: list of models (currently without their tops) :param datasets: list of dataset object :param Settings: settings :return: number of ready models

ModelHandler.ModelTester.**format_time**(*seconds*)

ModelHandler.ModelTester.**load_feature_file**(*path*)
Just loads the features stored in one file. :param path: :return:

ModelHandler.ModelTester.**load_features**(*filename_features_train*, *filename_features_test*, *y*, *y_val*)

ModelHandler.ModelTester.**predict_and_save_features**(*x*, *y*, *x_val*, *y_val*, *filename_features_train*, *filename_features_test*, *model*)

ModelHandler.ModelTester.**predict_from_generators**(*test_generator*, *val_generator*, *number_in_test*, *number_in_val*, *filename_features_train*, *filename_features_test*, *model*)

ModelHandler.ModelTester.**train_model**(*model*, *dataset*, *model_settings*)
Train model on a dataset using these settings :param model: model to be trained :param dataset: dataset to be used :param model_settings: model setting to be read for specifics :return:

ModelHandler.ModelTester.**train_models**(*models*, *datasets*, *Settings*)
Training on all models - dataset pairs from models. :param models: array of models to run :param dataset: datasets for the models :param Settings: settings which also describe which dataset belongs to which model :return:

ModelHandler.ModelTester.**train_top_model**(*model*, *model_settings*, *train_data*, *train_labels*, *validation_data*, *validation_labels*)
Train the whole model :param model: model with base and top :param model_settings: :param train_data: data for training :param train_labels: labels for training :param validation_data: data for validation (and validation error) :param validation_labels: labels for validation (and validation error) :return:

5.8 Module contents

CHAPTER 6

OSMHandler package

6.1 Submodules

6.2 OSMHandler.Checker module

OSMHandler.Checker.**Check**(*Segments*)

6.3 OSMHandler.ConnectionHandlerObj module

class OSMHandler.ConnectionHandlerObj.**ConnectionHandler**

Connection Handler

What does ConnectionHandler have?

- link to the database, set in initialization

What can it do?

- initialize connection to DB, prepare lists of columns we want to pay attention too
- query for a location

close_connection()

extract_all_pairs(*rows*, *colnames*, *excluded_column='dist_meters'*)

final_vec_invert_indices(*i*)

foo = -1

get_column_names_in_db()

load_key_attr_pairs(*csv_name*, *dont_take_keys_which_are_not_in_list*, *limit_number=-1*)

Builds the lists of interesting key-attribute pairs.

distinct_keys contains just the names of keys ex: [‘building’, ‘bridge’, ‘amenity’, . . .] We will load these columns out of the database.

list_of_watched_pairs contains ex: [‘building=yes’, ‘highway=residential’, ‘building=house’, . . .] We will count these occurances in the resulting neighborhood.

In list_of_watched_pairs we can find only combinations of key=attribute which have keys from distinct_keys!

In dont_take_keys_which_are_not_in_list we have column names of our available database, those show us, which keys we actually have about the data - so for any key not in there, we don’t have to make space in the final vector (as it would always be 0, for all the data).

Parameters

- **csv_name** – Name of tha file containing most common pairs of key-attribute combinations
- **limit_number** – Number of rows we would like to look at

Returns lists of distinct keys and key=attribute pairs we will pay attention to in sql query results

opened = False

query_location(*location*, *radius*, *table_name*)

report()

run_command(*command*)
Runs command and returns the rows and column names :param command: sql command :return:

setup_db_connection(*hostname*, *username*, *password*, *database*)
Establish connection

sql_cmd_everywhere(*column_names*, *table_name='planet_osm_line'*, *sql_limit_rows=-1*)

sql_cmd_radius(*column_names*, *table_name='planet_osm_line'*, *sql_limit_rows=-1*, *location=[]*, *radius=10*)

6.4 OSMHandler.Marker module

OSMHandler.Marker.**Mark**(*Segments*, *radius=50*, *interval=None*, *backwards=False*)
Mark Segments with radius. Call MarkSegment on each Segment. :param Segments: :param radius: radius in meters :return:

OSMHandler.Marker.**MarkSegment**(*Segment*, *radius=50*)
Mark Segment with new OSM vector depending on what the PosgreSQL db will tell us about the neighborhood. :param Segment: One Segment object initially without OSM data. :param radius: radius in meters :return:

OSMHandler.Marker.**MergeMarking**(*Segments1*, *Segments2*)

OSMHandler.Marker.**MergeMarking_LoadAndSave**(*pats_seg1*, *path_seg2*, *path_out*)

OSMHandler.Marker.**analyzeMarking**(*Segments*)
prints index boundaries of which segments are marked and which are not :param Segments: :return:

OSMHandler.Marker.**checkForStopFile**()

OSMHandler.Marker.**closeConnection**()

OSMHandler.Marker.**loadAndAnalyzeMarking**(*path*)

6.5 OSMHandler.TestDebugs module

6.6 Module contents

CHAPTER 7

Omnipresent module

Omnipresent.**array_md5**(*arr*)

Omnipresent.**file_exists_and_accesible**(*PATH*)

Omnipresent.**len_**(*L*)

Omnipresent.**save_job_report_page**(*folder_path*, *job_id*, *cut=True*)

Saves webpage generated by metacentrum just before the experiment run is finished. There wa can for example read the consumption of resources and time requirements.

Parameters

- **folder_path** –
- **job_id** – unique id provided by Metacentrum scheduling system. Links to the one unique page we want to get.
- **cut** – Shorten the output

Returns

Omnipresent.**send_mail**(*subject*, *message*, *attachment_path=None*)

Send mail! :param subject: Mail subject :param message: Mail message :param attachment_path: Login details and specifics of mails. :return:

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

e

m

o

Index

E

F

G

H

I

J

K

L

M

N

O

P

Q

U

V

X

Z