yolov8_dataset_merger package

Submodules

yolov8_dataset_merger.background_images module

yolov8_dataset_merger.background_images.background_images(*detector_classes: list[str], number_of_images: int, output_dir='C:\\Users\\FPepin\\Synology\\TwoWaysSync__TU_BERLIN_ProjektIAT_Robotergreifer\\perception\\venv\\Lib\\site-packages\\yolov8_dataset_merger\\background_images', instance_train_json_file='C:\\Users\\FPepin\\Synology\\TwoWaysSync__TU_BERLIN_ProjektIAT_Robotergreifer\\perception\\venv packages\\yolov8_dataset_merger\\instances_train2017.json'*) → None [source]

Function Summary

Downloads background images from the COCO dataset, excluding those that contain specified classes. The function saves the images and blank label files in a specified directory. These background images are intended for use in object detection models where they are used as negative samples (images without objects of interest, or backgroun images).

param detector_classes: A list of classes to exclude from the

background images. These are typically classes included in training, but in special cases, like in tomato detection where no class is excluded, the list can be empty. :type detector_classes: list[str] :param number_of_images: The number of background images to download. It is recommended to use between 0% and 10% background images relative to your dataset. :type number_of_images: int :param output_dir: The directory where the background images and corresponding blank label files will be saved. The default directory is “./background_images”. :type output_dir: str, optional :param instance_train_json_file : json file provided by coco on their website (https://cocodataset.org/#download) containing annotations of dataset. Path to this file. default value, place it in the script’s directory :type output_dir: str, not really optional, must be changed if file is changed :return: None :rtype: NoneType

Notes

- **The background images are saved to the *output_dir*, along with**

  corresponding blank *.txt* label files for each image.

- **The function uses the COCO API to retrieve image metadata and**

  downloads the images via their URLs.

Example

background_images([“cat”, “dog”], 100) This would download 100 images excluding those with “cat” and “dog”, and save them in the “./background_images” directory.

yolov8_dataset_merger.build_sphinx module

Module Summary

This module automates the generation of Sphinx documentation for a Python package using the ‘sphinx-apidoc’ command and provides utilities to clean up old documentation files before building the new documentation. The module also includes functionality to handle the execution of a Makefile (or make.bat on Windows) to build the documentation in different formats (e.g., HTML, LaTeX).

Functions

- ‘clean_rst_files(directory: str)’: Deletes all ‘.rst’ files in the specified directory, except for ‘index.rst’.
- ‘clean_build_directory(output_dir: str)’: Deletes all contents in the ‘_build’ directory, except the directory itself.
- ‘build_documentation(exclude_patterns: list[str], output_dir: str, source_dir: str)’: Generates Sphinx documentation using ‘sphinx-apidoc’ and builds the output in the specified format.
- ‘Makefile(output_dir: str, format: str = ‘html’)’: Runs the appropriate Makefile (or make.bat) to generate the documentation in the specified format.

Example

To run the script, ensure that *sphinx-apidoc* is installed and available in your system’s PATH, then call the *build_documentation()* function with your desired parameters:

```
exclude_patterns = ["*test*", ".vscode",
                    "datasets_to_merge", "merged_dataset*",
                    "runs", "instances_train2017.json",
                    "yolov8n-seg.pt", "yolov8n.pt"]
output_dir = r"C:\path\to\your\docs"
source_dir = r"C:\path\to\your\package-or-dir-containg-modules"
build_documentation(exclude_patterns, output_dir, source_dir)
```

Notes

- usefull primer tutorial
- ‘sphinx-apidoc’ must be installed.
- ‘sphinx’ must be configured in the output directory for building the documentation. This can be done with command ‘sphinx-quickstart’. Note that after configuration with the sphinx wizard, you’ll need to modify the conf.py file in output directory. For consistent style and behavior, configure as follow:

```python
# Configuration file for the Sphinx documentation builder.
#
# For the full list of built-in configuration values, see the documentation:
# https://www.sphinx-doc.org/en/master/usage/configuration.html
# -- Path setup --------------------------------------------------------------
import os
import sys
sys.path.insert(0, os.path.abspath('..'))
# -- Project information -----------------------------------------------------
# https://www.sphinx-doc.org/en/master/usage/configuration.html#project-information

project = 'YOLOv8 Dataset Merger'
copyright = '2024, Frederic Pepin'
author = 'Frederic Pepin'
release = '2024-09-30'

# -- General configuration ---------------------------------------------------
# https://www.sphinx-doc.org/en/master/usage/configuration.html#general-configuration

extensions = [
    'sphinx.ext.autodoc',
    'sphinx.ext.viewcode',
    'sphinx.ext.napoleon'
]

templates_path = ['_templates']
exclude_patterns = ['_build', 'Thumbs.db', '.DS_Store']

# -- Options for HTML output -------------------------------------------------
# https://www.sphinx-doc.org/en/master/usage/configuration.html#options-for-html-output

html_theme = 'sphinx_rtd_theme'
html_static_path = ['_static']
```

yolov8_dataset_merger.build_sphinx.Makefile(*output_dir, format: str = 'html'*) [source]

Function Summary

Execute Makefile independently of OS

param output_dir:	where documentation will be contained
type path_to_make_script:	str
param format:	format to export to, html is default. other formats are latexpdf, see sphinx documentation for necessary dependencies
type format:	str

yolov8_dataset_merger.build_sphinx.build_documentation(*exclude_patterns: list[str], output_dir: str = 'docs', source_dir='C:\\Users\\FPepin\\Synology\\TwoWaysSync__TU_BERLIN_ProjektIAT_Robotergreifer\\perception\\venv\\Lib\\site-packages\\yolov8_dataset_merger'*) [source]

Function Summary

Generates Sphinx documentation using the sphinx-apidoc command.

This function searches the current working directory for modules to be documented (by default) or given source folder. The output is saved to the specified directory (defaults to ‘docs’ in the current working directory). Specific file patterns can be excluded from the documentation generation.

param exclude_patterns:	A list of patterns to exclude from the documentation generation (e.g., test files, configuration folders, or non-module files).
type exclude_patterns:	list[str]

param output_dir:	The directory where the generated documentation will be saved (defaults to ‘docs’).
type output_dir:	str
param source_dir:	The directory of the source code (defaults to the current file’s directory).
type source_dir:	str
return:	None
rtype:	None

Notes

- **The function changes the working directory to the script’s location**

  before running the *sphinx-apidoc* command.

- The command output is captured for debugging purposes.
- in the index.rst file, *modules* should be appended:

```
 .. toctree::
:maxdepth: 2
:caption: Contents:
modules
[...]
```

yolov8_dataset_merger.build_sphinx.clean_build_directory(*output_dir: str*) [source]

Function Summary

Deletes all contents of the ‘_build’ directory except the director itself.

param output_directory:	The path to the output directory (docs) containing the ‘_build’ directory.
type output_directory:	str

yolov8_dataset_merger.build_sphinx.clean_rst_files(*directory: str*) [source]

Function Summary

Deletes all .rst files in the specified directory except ‘index.rst’.

param directory:	The directory to search for .rst files.
type directory:	str

yolov8_dataset_merger.change_class_according_to_HSV module

Module Summary

Module meant to change fruit class based on the color of the detected fruit.

Motivation: some datasets only identify tomatoes as such, without dif- ferentiating between ripe and unripe fruits. This is true for a spe- cial case, the TomatoOcclusion dataset, which addresses the problematic of occlusion of fruits by branches, leaves and other fruits, or a mix of all the latters.

In order to be compatible with other quality datasets (like TomatoLaboro) the need arises to characterize the ripeness of Tomatoes.

Based on the bounding box geometry and position, detect the presence of color spectrum (red) in a square box around center point of a bounding box (red is ripe). An enhanced approach would be to sample more regions in the bounding box, as it could be that a branch / leaf is present at middle point. But this also leads to the problem of a false detection of red of a ripe tomato behind an unripe one. Detection of partially hidden unripe tomatoes will also lead to problems if a red tomato overlaps the center of the former. Thus, the detection of occluded unripe tomato should be in my opinion left out. It is also less critical then the detection of ripe tomatoes in the bigger picture. Another stance could also be, falsely characterizing a unripe tomato is not critical, because unripe tomatoes continue to ripe also after harvest anyway. Therefore, the marginal phenomenon of characterizing an occluded unripe tomato as ripe because a red tomato overlaps ther region of interest at mid point doesn't have big repercussions and is likely to rarely happen in a given dataset.

yolov8_dataset_merger.change_class_according_to_HSV.change_class_based_on_color_all(*input_dir, output_dir='', ripe=2, unripe=4, ratio_center_box: int = 4*)→ None [source]

Function Summary

This function processes images of subdirectories "train, test, valid" in a given dataset directory to classify bounding boxes of tomatoes based on their ripeness by analyzing the color in a Region of Interest (ROI) around each bounding box's center point. It updates the class IDs in the label files based on the ripeness of each detected tomato.

The function reads all images and their corresponding label files in the specified directory, checks the ripeness of the tomatoes inside the bounding boxes using HSV color values, and updates the class ID in the label files accordingly. Additionally, it generates copies of the images with bounding boxes drawn on them for visual verification.

param input_dir:	Path to the input directory containing folders of images and labels.
type input_dir:	str
param output_dir:	Path to the output directory where the modified label files and image copies will be saved. if value is left out, default behavior creates another directory in the parent dataset folder called "image_copies_with_bounding_boxes"
type output_dir:	str
param ripe:	The class ID numberto assign to ripe tomatoes. Default is irrelevant, must be changed to fit a given structure in main .yaml file
type ripe:	int, optional
param unripe:	The class ID to assign to unripe tomatoes. Default is irrelevant, must be changed to fit a given structure in main .yaml file
type unripe:	int, optional
param ratio_center_box:	Ratio that defines the size of the sampling region for checking ripeness, relative to the bounding box size. Default is 4.
type ratio_center_box:	int, optional
return:	None
rtype:	NoneType

yolov8_dataset_merger.change_class_according_to_HSV.check_tomato_ripeness(*image, x_normalized, y_normalized, box_width_norm, box_height_norm, ratio_center_box: int = 4*)→ bool [source]

Functíon Summary

Determines if a tomato within a bounding box in the image is ripe by evaluating the color within the bounding box in the HSV color space. The image is sampled around the center of the bounding box, and the HSV values of the region are analyzed to check for the presence of red, indicating ripeness.

The function takes normalized coordinates for the center of the bounding box and normalized box dimensions, converts them to pixel coordinates, and samples a region around the center. The region is converted to the HSV color space, and the ripeness is checked by determining if any pixels fall within the red color range.

param image:	Input image in BGR color space
type image:	np.ndarray
param x_normalized:	Normalized x-coordinate for the center of the bounding box.
type x_normalized:	float
param y_normalized:	Normalized y-coordinate for the center of the bounding box.
type y_normalized:	float
param box_width_norm:	Normalized width of the bounding box, as a proportion of the image width.
type box_width_norm:	float
param box_height_norm:	Normalized height of the bounding box, as a proportion of the image height.
type box_height_norm:	float
param ratio_center_box:	Ratio that determines the size of the sample region in relation to the box dimensions. The default value is 4, meaning the sample region will be 1/4th the size of the box.
type ratio_center_box:	int, optional
return:	A boolean indicating whether the tomato in the region is ripe (True) or unripe (False).
rtype:	bool

---

yolov8_dataset_merger.change_class_according_to_HSV.draw_bounding_boxes(*image: ndarray, bounding_boxes: list[list[str]], ratio_center_box: int = 4*)→ ndarray [source]

Function Summary

Draws bounding boxes around objects (tomatoes) in an image based on the provided bounding box coordinates, which are in a normalized format. In addition to the main bounding box, the function draws a smaller square box around the center of each object, sized proportionally to the original bounding box (longest edge). This smaller box is used for further analysis, such as checking for color or ripeness.

green rectangles for the main bounding boxes and blue rectangles for the smaller boxes around the center points.

param image:	The input image on which the bounding boxes will be drawn. This image is typically in BGR color space.
type image:	np.ndarray
param bounding_boxes:	A list of bounding boxes, where each bounding box is a list containing the class ID, normalized x and y center coordinates, and normalized width and height of the box.
type bounding_boxes:	list of lists of strings
param ratio_center_box:	The ratio that defines the size of the smaller box relative to the main bounding box. The default value is 4, meaning the smaller box will be 1/4th the size of the original bounding box.
type ratio_center_box:	int, optional
return:	A copy of the input image with the bounding boxes drawn on it.
rtype:	np.ndarray

Notes

- **The normalized coordinates for the bounding boxes are converted**

  to pixel coordinates based on the dimensions of the input image.

- **The smaller box around the center point is square, and its size**

  is determined by the larger of the two dimensions (width or height) of the main bounding box.

---

yolov8_dataset_merger.change_class_according_to_HSV.get_pixel_coordinates(*x_normalized: float*, *y_normalized: float*, *image_width: float*, *image_height: float*) → tuple[float] [source]

Function Summary

compute absolute geometry in pixel sizes out of normalized yolov8 coordinates

param x_normalized:	
type x_normalized:	float
param y_normalized:	
type y_normalized:	float
param image_width:	absolute image width
type image_width:	float
param image_height:	absolute image height
type image_height:	float
return:	tuple of (x_pixel, y_pixel), absolute x and y values in the coordinate system of the image
rtype:	tuple[float]

---

yolov8_dataset_merger.change_class_according_to_HSV.is_ripe_tomato(*hsv_region: ndarray*) → bool [source]

Function Summary

Determines if a given Region of Interest (hsv_region) contains any red pixels, indicating ripeness. In this case, hsv_region is a space containing the hsv values of each pixel in a given geometrical region.

Each pixel in the HSV region is checked to see if its HSV values fall into either of these ranges. If a pixel's values fall into one of these ranges, the corresponding mask will have a 1 (white) at that pixel's location. If not, the mask will have a 0 (black).

param hsv_region:	image is first transformed to a color space with cv2.cvtColor method, and a subspace is created for the Region of Interest at center point, which is hsv_region. here, has 3 axes
type hsv_region:	np.array
return:	ripe (True) or unripe (False)
rtype:	bool

yolov8_dataset_merger.common_functions module

Module Summary

This module contains utility functions for working with file directories and manipulating nested lists. It includes functions to flatten a list of lists, create unique directories, change directory permissions, and recursively find directories containing specific subfolders ("images" and "labels").

Functions

- **flatten_list: Flattens a nested list into a single one-dimensional**

  list.

- mkdir_unique: Creates a unique directory at a specified path.
- **change_permissions: Changes the permissions of a directory and its**

  contents recursively.

- **find_images_labels_folders: Searches for directories containing**

  “images” or “labels” subfolders.

- **count_number_ofimage_and_labels: counts the number of images and**

  labels contained in a dataset (train, test and val all comprised together)

Example

```
>>> from my_module import flatten_list, mkdir_unique,
    find_images_labels_folders
>>> flattened = flatten_list([[1, 2, [3]], [4, 5]])
>>> print(flattened)
[1, 2, 3, 4, 5]
>>> dir_path = mkdir_unique("my_directory")
>>> folders = find_images_labels_folders("/path/to/data")
>>> print(folders)
['/path/to/data/folder1', '/path/to/data/folder2']
```

---

yolov8_dataset_merger.common_functions.change_permissions(*directory*, *mode=511*) [source]

---

yolov8_dataset_merger.common_functions.count_number_of_image_and_labels(*dataset_path: str*)
[source]

Function Summary

Counts the total number of images and label files in a dataset structured with subdirectories containing “images” and “labels” folders. Usefull for debugging and making sure merges have occurred without errors

param dataset_path:	The path to the dataset directory which contains subdirectories with “images” and “labels” folders.
type dataset_path:	str
return:	A tuple containing two integers: the total number of images and the total number of label files found in the dataset. (number_images,number_labels)
rtype:	tuple[int, int]

Example

```
>>> total_images, total_labels = count_number_of_image_and_labels("path/to/dataset")
>>> print(total_images, total_labels)
(150, 150)
```

Notes

- **The function assumes that each subdirectory contains a folder named**

  "images" and a folder named "labels". If these folders are not present, the function will skip that subdirectory.

---

yolov8_dataset_merger.common_functions.find_images_labels_folders(*input_dir*) [source]

Function Summary

Recursively searches through a directory to find subfolders named "images" or "labels". When such folders are found, the function adds the parent directory containing these subfolders to a list.

This function is designed to locate directories that contain images and labels for further processing in a dataset.

param dir:	The path of the directory to search for subfolders. The search is performed recursively within all nested subdirectories.
type dir:	str
return:	A list of directories that contain either an "images" or "labels" subfolder.
rtype:	list

Example

```
>>> folders = find_images_labels_folders_nested("/path/to/data")
>>> print(folders)
['/path/to/data/folder1', '/path/to/data/folder2']
```

---

yolov8_dataset_merger.common_functions.flatten_list(*self, list_of_lists: list*) [source]

Function Summary

Flattens a list containing nested lists into a single one-dimensional list. If the input is not a list, it returns a list containing the input element.

param list_of_lists:	A list that may contain nested lists. The function will recursively flatten any nested structures within it.
type list_of_lists:	list
return:	A one-dimensional list containing all the elements from the input list, including those from any nested lists.
rtype:	list

Example

```
>>> flatten_list([[1, 2, [3]], [4, 5]])
[1, 2, 3, 4, 5]
```

```
>>> flatten_list(10)
[10]
```

---

yolov8_dataset_merger.common_functions.mkdir_unique(*new_dir_path*) [source]

Function Summary

Creates a new directory at the specified path. If the directory already exists, a unique directory name is generated by appending an index to the base name, and the new directory is created.

param new_dir_path:	The path where the new directory should be created. If a directory with this name already exists, a unique name will be generated.
type new_dir_path:	str
return:	The path to the newly created directory, whether it is the original or the unique one.
rtype:	str

Example

```
>>> mkdir_unique("my_directory")
'my_directory'  # If it didn't exist initially
```

```
>>> mkdir_unique("my_directory")
'my_directory_1'  # If 'my_directory' already existed
```

yolov8_dataset_merger.convert_segmentation_mask_to_bounding_box module

Module Summary

This module provides functionalities to process segmentation masks from files, converting them into bounding boxes that conform to the YOLO format. It includes functions for reading masks from text files, computing bounding box coordinates, and writing the results to an output directory.

Functions

- convert_mask_to_boundung_box(mask: str) -> str
- **process_masks_to_bounding_boxes(input_dir: str,**

  output_dir: str) -> None

Usage

To convert mask files to bounding box format, use the 'process_masks_to_bounding_boxes' function, providing the input and output directory paths.

---

yolov8_dataset_merger.convert_segmentation_mask_to_bounding_box.convert_mask_to_boundung_box(*mask: str*) [source]

Function Summary

Converts a segmentation mask from a string format to a bounding box in YOLO format.

The segmentation mask directly read from the annotation file is a single string, whose values are separated by blank spaces. This function first transforms this string into a list of substrings. The first integer value is the class number, followed by the {x1, y1, x2, y2, ..., xn, yn} coordinates

pairs of the segmentation points. By looping through those values, we can determine the extrema of the mask on both axes and define a bounding box in YOLO format, namely: [class_number, x_center, y_center, width, height].

param mask:	Mask directly extracted from .txt file.
type mask:	str
return:	A string representing the bounding box in YOLO format: [class_number (int), x_center (float), y_center (float), width (float), height (float)].
rtype:	str

Example

```
>>> convert_mask_to_boundung_box("1 50 50 100 100")
"1 75.0 75.0 50.0 50.0
"
```

---

yolov8_dataset_merger.convert_segmentation_mask_to_bounding_box.process_masks_to_bounding_boxes(*input_dir: str, output_dir: str*) [source]

Function Summary

Processes all mask files in the input directory, converting each mask to a bounding box format and saving the results to the specified output directory. Each mask is read from text files, and the bounding boxes are calculated using the convert_mask_to_boundung_box function.

param input_dir:	The directory containing subdirectories of mask files to be processed.
type input_dir:	str
param output_dir:	The directory where the resulting bounding boxes will be saved.
type output_dir:	str
return:	None

---

yolov8_dataset_merger.convert_segmentation_mask_to_bounding_box.rund(*number, digits=5*) [source]

yolov8_dataset_merger.examples module

Module Summary

Module to show case some miscellaneous pieces of code. Functions are nevertheless usable

---

yolov8_dataset_merger.examples.coco_to_yolo() [source]

Example transform COCO Dataset in YOLOv8

```
import ultralytics.data.converter as u
u.convert_coco(r"TomatoLaboro\annotations",r".\output_labels",True)
```

---

yolov8_dataset_merger.examples.main(*path_to_all_datasets_folder: str*) [source]

Dataset Merger Code Example

```
import os
import time
from resolve_filenames_conflicts import merge_datasets
tic = time.time()
path_to_all_datasets_folder = r"path/to/folder/containing/all/datasets"
# output in the script's directory
output_path = os.path.dirname(__file__)
merge_datasets(path_to_all_datasets_folder,output_path)
toc = time.time()
print(f"execution time: {toc-tic:0.4f} seconds")
```

param path_to_all_datasets_folder: _description_
type path_to_all_datasets_folder: str

---

yolov8_dataset_merger.examples.train_model(*dataset_path: str, model*) [source]

Train Model Example Code

```
from ultralytics import YOLO
# based on a pre-trained yolo omdel
model = YOLO("yolov8n-seg.pt")
results = model.train(data = r"path/to/dataset",epochs=100)
```

param dataset_path: _description_
type dataset_path: str
param model: pre trained model, YOLO("yolov8n-seg.pt")
type model: _type_, optional

yolov8_dataset_merger.map_classes_number module

Module Summary

This module provides functions to remap class integers in label files based on mappings defined in a YAML files. YAML files are expected to have been previously automatically remapped with the module *yaml_file_merger* to provide for a clean remapping of each classes ids. It is useful when merging datasets that may have conflicting or different class numbering/ids.

Functions

- replace_class_number_in_file: Remaps class integers in label files based on a mapping provided in a list of tuples.
- replace_remapped_class_number: Replaces class integers in label files based on a mapping defined in a remapped YAML file.
- replace_class_number_from_folder: Replaces class integers in all label files found within a specified folder containing multiple datasets.

Usage

This module is intended to be used for preprocessing label files before merging datasets to ensure consistent class numbering.

yolov8_dataset_merger.map_classes_number.replace_class_number_from_folder(*folder_containing_datasets: str, remapped_yaml_file_path: str, replace=True*) [source]

Function Summary

Replaces class integers in all label files found within a specified folder containing multiple datasets. The class integers are updated based on the mapping defined in a given remapped YAML file. (see module yaml_file_merger)

param folder_containing_datasets:	The path to the folder that contains multiple datasets, each having their own label files.
type folder_containing_datasets:	str
param remapped_yaml_file_path:	Path to the YAML file that defines the remapped class integers.
type remapped_yaml_file_path:	str
param replace:	If True, replaces the original labels with remapped ones and deletes the originals. Default is True.
type replace:	bool
return:	None

yolov8_dataset_merger.map_classes_number.replace_class_number_in_file(*labels_directory: str, original_new_tuple_list: list[tuple], replace=False, output_folder_name: str = 'mapped_labels'*) [source]

Function Summary

Remaps class integers in label files based on a mapping provided in a list of tuples. This is useful when merging datasets to ensure

param labels_directory:	Path to the directory containing label files to be remapped.
type labels_directory:	str
param original_new_tuple_list:	List of tuples where each tuple contains an old class integer and its corresponding new class integer. (old,new)
type original_new_tuple_list:	list[tuple[int]]
param replace:	If True, replaces the original labels directory with the remapped one and deletes the original. Default is False.
type replace:	bool
param output_folder_name:	Name of the output folder for the remapped labels. If not provided, it will be created in the same parent directory as labels_directory. Default is “mapped_labels”.
type output_folder_name:	str
return:	None

Example

```
>>> replace_class_number_in_file("path/to/labels",
>>>                              [(0, 1), (1, 2), (2, 3)])
```

yolov8_dataset_merger.map_classes_number.replace_remapped_class_number(*remapped_yaml_file_path: str, labels_directories: list[str], replace=False, output_folder_name: str = 'mapped_labels'*) [source]

Function Summary

Replaces class integers in label files based on a mapping defined in a remapped YAML file (see module yaml_file_merger). For datasets with different class numberings.

param remapped_yaml_file_path:	Path to the YAML file that defines the remapped class integers.

type remapped_yaml_file_path:	str
param labels_directories:	List of paths to directories containing label files to be updated.
type labels_directories:	list[str]
param replace:	If True, replaces the original labels with remapped ones and deletes the originals. Default is False.
type replace:	bool
param output_folder_name:	Name of the output folder for remapped labels. If not provided, it will be created in the same parent directory as labels_directory. Default is “mapped_labels”.
type output_folder_name:	str
return:	None

yolov8_dataset_merger.resolve_filenames_conflicts module

Module Summary

This module provides functionality to merge multiple datasets that cont- ain images and corresponding label files. The datasets are expected to be structured in subdirectories for training, testing, and validation, with each containing their respective ‘images’ and ‘labels’ directories.

The module includes functions for resolving filename conflicts during the merge process, as well as generating a consolidated YAML file that contains class remapping information.

Functions

1. **resolve_filenames_conflicts_and_merge(path_list: list[str],**

   output_name=None, base_directory=None) -> None

   - Merges multiple directories containing images and labels, resolving filename conflicts by appending suffixes to duplicate filenames.
2. **merge_datasets(path_to_folder_containing_datasets: str,**

   output_dataset_path: str = None) -> None

   - Merges datasets from a specified folder into a single dataset, organizing the output into subdirectories for training, testing, and validation. Creates a merged YAML file containing class remapping information.

Dependencies

- *yaml_file_merger*: For merging YAML files.
- *map_classes_number*: For remapping class identifiers in label files.
- *common_functions*: For utility functions such as creating unique directories.

Notes

Ensure that the datasets being merged follow the expected directory structure for the module to function correctly. Typically, this means that a yolov8 dataset folder contains a .yaml file containing metadata, subfolders called “train”, “test” and “val” or “valid”. These subfolders have their own subfolders that should be called “images” and “labels”. Any discrepancy in the names will result in failure of the functions.

yolov8_dataset_merger.resolve_filenames_conflicts.are_there_conflicts(*folder: str*) [source]

summary returns a dictionary of file names that occur more than once in the given directory, with filenames as key and number of occurences as value :param folder: path/to/folder :type folder: str

Returns: dict

yolov8_dataset_merger.resolve_filenames_conflicts.merge_datasets(*path_to_folder_containing_datasets: str, output_dataset_path: str = None*) [source]

Function Summary

Merges multiple datasets contained within a specified folder into a single dataset. This function organizes the merged dataset into subdirectories for training, testing, and validation. It creates a merged .yaml file which contains metadata, most importantly class ids and their remapped numbering, with the submodule *yaml_file_merger*. It will then modify the label files to change the class ids according the the remapping in the yaml file, with submodule *map_classes_number*.

param path_to_folder_containing_datasets:	The path to the folder that contains the individual datasets with t respective training, testing, and validation folders, and their yaml
type path_to_folder_containing_datasets:	str
param output_dataset_path:	The optional path where the merged dataset will be saved. If not provided, the merged dataset will be created in the same parent d as the input datasets.
type output_dataset_path:	str, optional
return:	None

Example

```
>>> merge_datasets("path/to/folder/containing/datasets")
```

yolov8_dataset_merger.resolve_filenames_conflicts.resolve_filenames_conflicts_and_merge(*path_list: list[str], output_name=None, base_directory=None*) [source]

Function Summary

Merges multiple directories that contain subdirectories named “images” and “labels”. The function resolves filename conflicts by appending a suffix to conflicting ones. This is useful for consolidating datasets while maintaining unique identifiers.

:param : must contain subdirectories named “images” and “labels”. Those

directories are typically “train”, “test” and “val” or “valid”, for yolov8

:type : param path_list: A list of directories to be merged. Each directory :param : :type : type path_list: list[str] :param : provided, a default output directory will be created. This is on the

level of “train”, “test” and “valid”

:type : param output_name: The name of the output directory. If not :param : :type : type output_name: str, optional :param : directory will be stored. It is the global output merged dataset

directory. If not provided, it will default to the parent directory of the first images directory.

:type : param base_directory: The base directory where the output :param : :type : type base_directory: str, optional :param : :type : return: None

yolov8_dataset_merger.supervisely_json_to_yolov8 module

Module Summary

module containing functions to transform JSON Supervisely annotations of a dataset to YOLOv8 annotations.

---

yolov8_dataset_merger.supervisely_json_to_yolov8.json_syntax_to_yolov8_all(*input_dir: str, output_dir: str = ''*)→ None [source]

Function Summary

call the parsing function on all files contained in a given input direc- tory, write the YOLOv8 formatted data to a .txt file in output directory

param input_dir:	path to directory containing supervisely annotations
type input_dir:	str
param output_dir:	(optional) path to output directory, where to store the yolov8 annotations. default behavior, creates a directory called 'yolov8_annoations' in the parent directory of input_dir
type output_dir:	str

---

yolov8_dataset_merger.supervisely_json_to_yolov8.json_syntax_to_yolov8_file(*file_path: str*)→ list[str] [source]

Function Summary

takes a supervisely file (JSON format), parses data contained in file, and transforms to a yolov8 format.

param file_path:	
type file_path:	str
return:	list of all bounding boxes, each bounding box is a string with format '<class ID> <normalized x_center> <normalized y_center> <normalized box width> <normalized box height>n'
rtype:	list[str]

---

yolov8_dataset_merger.supervisely_json_to_yolov8.rund(*number: float, digits=6*) [source]

yolov8_dataset_merger.yaml_file_merger module

Module Summary

This module provides functions to parse and merge YAML files that contain class mappings for different datasets. It ensures that improper formatting of yaml files is handled, remapping class names to unique integers and maintaining a count of the total classes. This functionality is especially useful when combining datasets that may have conflicting or different class numbering.

Functions

- parse_yaml_file: Parses a YAML file, converting the “names” key into a dictionary format and ensuring the file’s content is valid for further processing.
- merge_yaml_files: Merges multiple YAML files into a single YAML file, remapping class names to unique class integers and updating the total class count.
- merge_yaml_files_from_folder: Merges all YAML files found within subdirectories of a specified folder, compiling their contents and remapping class names to unique integers.

---

yolov8_dataset_merger.yaml_file_merger.merge_yaml_files(*yaml_files_paths_list: list[str], output_path=None*) [source]

Function Summary

Merges multiple YAML files that contain class mappings into a single YAML file. This function ensures that class names are remapped to unique class integers and updates the total class count in the output.

param yaml_files_paths_list:	A list of paths to the YAML files that need to be merged.
type yaml_files_paths_list:	list[str]
param output_path:	The optional path where the merged YAML file will be saved. If not provided, it defaults to the parent directory of the first dataset.
type output_path:	str, optional
return:	The path to the merged YAML file that was created.
rtype:	str

Example

```
>>> merged_path = merge_yaml_files(["path/to/dataset1/data.yaml",
...                                 "path/to/dataset2/data.yaml"])
>>> print(merged_path)
"path/to/parent/merged_data.yaml"
```

Notes

- **The function assumes that each YAML file contains a “names” key**

  mapping class integers to class names.

- **If a class name already exists in the output dictionary, it**

  will not be added again.

- **The total number of classes (‘nc’) is updated in the output**

  YAML file based on the unique class names.

---

yolov8_dataset_merger.yaml_file_merger.merge_yaml_files_from_folder(*folder_containing_datasets: str, output_path=None*) [source]

Function Summary

Merges all YAML files found within subdirectories of a specified folder into a single YAML file. The function searches for the first YAML file in each dataset folder and compiles their contents while remapping class names to unique integers.

param folder_containing_datasets:	The path to the folder that contains multiple dataset subdirectories, each expected to have a YAML file with class mappings.
type folder_containing_datasets:	str
param output_path:	The optional path where the merged YAML file will be saved. If not provided, it defaults to the parent directory, namely folder containing all subdatasets.
type output_path:	str, optional
return:	The path to the merged YAML file that was created.
rtype:	str

Example

```
>>> merged_yaml_path = merge_yaml_files_from_folder("path/to/dataset_folder")
>>> print(merged_yaml_path)
"path/to/parent/merged_data.yaml"
```

Notes

- **The function expects each subdirectory to contain at least one**

  YAML file. If no YAML file is found, the corresponding folder will be skipped.

- **The merged YAML file will contain unique class mappings from**

  all the input YAML files.

---

yolov8_dataset_merger.yaml_file_merger.parse_yaml_file(*yaml_file: str*) [source]

Function Summary

Parses a YAML file by removing any tab characters and converting the names specified in the file into a dictionary format. This function ensures that the resulting content is valid for further processing.

param yaml_file:	The file path to the YAML file that needs to be parsed.
type yaml_file:	str
return:	A dictionary containing the parsed content of the YAML file. The “names” key, if initially a list, is transformed into a dictionary where the index is the key and the corresponding name is the value.
rtype:	dict

Example

```
>>> parsed_content = parse_yaml_file("path/to/yaml_file.yaml")
>>> print(parsed_content)
{'names': {0: 'class_1', 1: 'class_2', ...}}
```

Notes

- **This function replaces tab characters with double spaces since**

  tabs are not allowed in YAML syntax.

Module contents