PV4GER

Release 0.0.1

Kevin Mayer

May 24, 2021

Table of Contents

PV4GER aims at democratizing and accelerating the access to photovoltaic (PV) systems data in Germany and beyond. To do so, we have developed a computer vision-based pipeline which leverages 3D building data to automatically create address-level PV registries for all counties within Germany's most populous state North Rhine-Westphalia.

For a detailed description of the underlying pipeline and a case study for the city of Bottrop, please have a look at our spotlight talk *"An Enriched Automated PV Registry: Combining Image Recognition and 3D Building Data"* at NeurIPS 2020:

- Paper
- Slides
- Recorded Talk

To execute the pipeline for a given county, make sure that you have:

1. Downloaded the pre-trained model weights for PV classification and segmentation from our public S3 bucket.
2. Set up and activated your local virtual environment based on the provided *requirements.txt*.
3. Downloaded the .GeoJSON with all the rooftop information for your selected county from our public S3 bucket.
4. Set up the config.yml appropriately before executing *run_pipeline.py*

Chapter 1

Pipeline Configuration

The only parts in **config.yml** that you need to touch are:

bing_key:
Put your Bing API key here. You only need a Bing API key if you intend to run *registry_creator.py*. The API key is needed to translate the latitude and longitude values of detected PV systems into actual street addresses.

county4analysis:
Specify the county in North Rhine-Westphalia for which you want to run the analysis by name, e.g. *Essen*.

run_tile_creator:
Put 1 if you would like to execute this pipeline step and 0 if you would like to skip it.

run_tile_downloader:
Put 1 if you would like to execute this pipeline step and 0 if you would like to skip it.

run_tile_processor:
Put 1 if you would like to execute this pipeline step and 0 if you would like to skip it.

run_tile_coords_updater:
Put 1 if you would like to execute this pipeline step and 0 if you would like to skip it.

run_registry_creator:
Put 1 if you would like to execute this pipeline step and 0 if you would like to skip it.

Pipeline Step 1: Tile Creator

class src.pipeline_components.tile_creator.**TileCreator** (*county_handler*)

Class to generate a list specifying all tiles within a given county by their minx, miny, maxx, maxy coordinates.

Variables

- **output_path** (*Path*) – Path to the pickle file which saves the list of all tiles within the selected county.
- **radius** (*int*) – Earth radius in meters.
- **side** (*int*) – Side length in meters for the tiles.
- **N** (*float*) – Northern boundary for the tile coordinates.
- **S** (*float*) – Southern boundary for the tile coordinates.
- **E** (*float*) – Eastern boundary for the tile coordinates.
- **W** (*float*) – Western boundary for the tile coordinates.
- **polygon** (*shapely.geometry.polygon.Polygon*) – Geo-referenced polygon geometry for the selected county within NRW.

defineTileCoords ( )

Spans a grid of tiles, each with a dimension 240m x 240m, over North Rhine-Westphalia and saves the tiles within the respective county by their minx, miny, maxx, maxy coordinates. Only tiles where at least one corner is within the county's polygon will be saved and later downloaded.

Pipeline Step 2: Tile Downloader

class src.pipeline_components.tile_downloader.**TileDownloader** (*configuration*, *polygon*, *tile_coords*)

Class to download tiles from the openNRW web server in a multi-threaded fashion.

Variables

- **tile_dir** (*Path*) – Path to directory where all the downloaded tiles are saved.
- **downloaded_path** (*Path*) – Specifies the path to the document which saves all the tiles by their minx, miny, maxx, maxy coordinates which were successfully downloaded.
- **not_downloaded_path** (*Path*) – Specifies the path to the document which saves all the tiles by their minx, miny, maxx, maxy coordinates which were **not** successfully downloaded.
- **WMS_1** (*str*) – Initial URL stub for requests to the openNRW server.
- **WMS_2** (*str*) – Final URL stub for requests to the openNRW server.
- **NUM_THREADS** (*int*) – Number of threads used to simultaneously download tiles from the openNRW server.

download (*Tile_coords*, *threadCounter*)

Download tiles from openNRW's web servers.

Parameters

- **Tile_coords** (*list*) – List of tuples. Each tuple specifies a to be downloaded tile by its minx, miny, maxx, maxy.
- **threadCounter** (*int*) – ID to distinguish between the different threads working in parallel.

Pipeline Step 3: Tile Processor

class src.pipeline_components.tile_processor.**TileProcessor** (*configuration*, *polygon*)

Class which splits tiles into smaller images, performs a binary classifiaction on each image to identify PV panels and segments a PV system's area on positively classified images.

Variables

- **cls_threshold** (*float*) – Threshold value with respect to the classification network's softmax score above which an image is classified as positive.
- **seg_threshold** (*float*) – Threshold value to turn the segmentation model's final class activation maps into binary segmentation masks.
- **batch_size** (*float*) – Specifies the number of samples per batch processed by the classification network.
- **input_size** (*float*) – Specifies the dimension of the images being processed by the classification network. This should not be changed.
- **cls_checkpoint_path** (*str*) – Path for loading the pre-trained classification weights.
- **seg_checkpoint_path** (*str*) – Path for loading the pre-trained segmentation weights.
- **tile_dir** (*str*) – Path to directory where all the downloaded tiles are saved.
- **pv_db_path** (*Path*) – Path to the .csv file which saves the tile ID, the image ID, and the geo-referenced polygon for all identified PV systems.
- **processed_path** (*Path*) – Path to the .csv file which saves all the tile IDs which have been successfully processed.
- **not_processed_path** (*Path*) – Path to the .csv file which saves all the tile IDs which have been **not** successfully processed.
- **cls_model** (*torchvision.models.inception.Inception3*) – Model to identify PV panels on aerial imagery.
- **seg_model** (*torchvision.models.segmentation.deeplabv3.DeepLabV3*) – Model to segment PV panels on aerial imagery.
- **dataset** (*src.dataset.dataset.NrwDataset*) – All the images which will be processed by our PV pipeline.
- **polygon** (*shapely.geometry.polygon.Polygon*) – Geo-referenced polygon geometry for the selected county within NRW.
- **radius** (*int*) – Earth radius in meters.
- **side** (*int*) – Aerial image side length in meters.
- **size** (*int*) – Aerial image side length in pixels.
- **dlat** (*float*) – Spans a distance of 16 meters in north-south direction.

- **polygonCreator** (*src.utils.polygon_creator.PolygonCreator*) – Turns binary segmentation mask of PV systems into geo-referenced polygons.

run ( )

Loads dataset of tiles, splits each tile into 16m x 16m images, and processes the aerial images within the specified county by detecting and segmenting PV panels.

Optional Step: Tile Updater

class src.pipeline_components.tile_updater.**TileCoordsUpdater** (*configuration=None*, *tile_coords=None*)

In case the tile processing is halted or aborted, this class can be used to update the list of tiles by removing all the already processed tiles.

Variables

- **old_tile_coords** (*list*) – List of tuples with all the tiles within the selected county.
- **county** (*str*) – The name of the county for which you run the analysis.
- **tile_coords_path** (*Path*) – Path to the pickle file which stores the list of tuples for all tiles within a given county.
- **processed_path** (*Path*) – Path to the .csv file which stores all the already processed tiles within a given county.

update ( )

Removes all the already processed tiles within a given county from the list of tiles which ought to be processed.

Pipeline Step 4: Registry Creator

class src.pipeline_components.registry_creator.**RegistryCreator** (*configuration*)

Creates an address-level and rooftop-level PV registry for the specified county by bringing together the information obtained from the tile processing step with the county's 3D rooftop data.

Variables

- **county** (*str*) – Name of the county for which the enrichted automated PV registry is created.
- **raw_PV_polygons_gdf** (*GeoPandas.GeoDataFrame*) – Contains all the identified and segmented PV panels within a given county based on the results from the previous tile processing step.
- **rooftop_gdf** (*GeoPandas.GeoDataFrame*) – Contains all the rooftop information such as a rooftop's tilt, its azimuth, and its geo-referenced polygon derived from openNRW's 3D building data.
- **bing_key** (*str*) – Your Bing API key which is needed to reverse geocode lat, lon values into actual street addresses.
- **corrected_PV_installations_on_rooftop** (*GeoPandas.GeoDataFrame*) – GeoDataFrame with preprocessed PV polygons matched to their respective rooftop segments

adjust_detected_pv_area_by_tilt (*raw_PV_installations_on_rooftop: Optional[geopandas.geodataframe.GeoDataFrame] = None*) → geopandas.geodataframe.GeoDataFrame

Adjusts detected PV area by considering the rooftop's tilt

Parameters **raw_PV_installations_on_rooftop** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which must contain the columns "Tilt" and "area_inter"

Returns Input GeoDataFrame extended by an additional column named "area_tilted" which adjusts the detected PV area by considering the rooftop's tilt

Return type GeoPandas.GeoDataFrame

aggregate_raw_PV_polygons_to_raw_PV_installations (*raw_PV_polygons_gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Aggregate raw PV polygons belonging to the same PV installation. Raw refers to the fact that the PV area is not corrected by the tilt angle. For each PV installation, we compute its raw area and a unique identifier.

Parameters **raw_PV_polygons_gdf** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which contains all the raw PV polygons which have been detected during the previous pipeline step.

Returns GeoDataFrame with dissolved PV polygon geometries.

Return type GeoPandas.GeoDataFrame

append_raw_overhanging_PV_installations_to_intersected_installations (*raw_overhanging_PV_installations: Optional[geopandas.geodataframe.GeoDataFrame] = None*, *raw_PV_installations_on_rooftop: Optional[geopandas.geodataframe.GeoDataFrame] = None*) → geopandas.geodataframe.GeoDataFrame

PV polygons which do not intersect with a rooftop polygon, although they do border to a rooftop, are matched to their nearest rooftop geometry and appended to the GeoDataFrame listing all rooftop PV polygons

Parameters
- **raw_overhanging_PV_installations** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which specifies all the PV polygons which border to a rooftop, but are not intersected with a rooftop geometry
- **raw_PV_installations_on_rooftop** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which specifies all the PV polygons which are intersected with a rooftop geometry

Returns GeoDataFrame where overhanging PV installations have been enriched with the attributes of the closest rooftop and appended to raw_PV_installations_on_rooftop

Return type GeoPandas.GeoDataFrame

calculate_distance_in_meters_between_raw_overhanging_pv_installation_centroid_and_ne (*raw_overhanging_pv_installations_enriched_with_closest_rooftop_data: Optional[geopandas.geodataframe.GeoDataFrame] = None*)

Calculate the distance in meters between the centroid of the overhanging PV polygon, here points_no_data, and the PV polygon centroid which is intersected with a rooftop polygon, here address_points

Parameters **raw_overhanging_pv_installations_enriched_with_closest_rooftop_data** (*GeoPandas.GeoDataFrame*) – GeoDataFrame where overhanging PV installations have been enriched with the attributes of the closest rooftop

Returns GeoDataFrame where overhanging PV installations have been enriched with the attributes of the closest rooftop with an additional attribute which specifies the distance between the centroid of the overhanging PV polygon and the centroid of the intersected PV polygon in meters

Return type GeoPandas.GeoDataFrame

calculate_pv_capacity (*registry: Optional[geopandas.geodataframe.GeoDataFrame] = None*) → geopandas.geodataframe.GeoDataFrame

Converts the detected PV area into a PV capacity estimate

Parameters **registry** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which must contain the columns “area_inter” and “area_tilted”

Returns Input GeoDataFrame extended by two additional columns named “capacity_not_tilted_area” and “capacity_tilted_area”

Return type GeoPandas.GeoDataFrame

clip_incorrect_tilts (*raw_PV_installations_on_rooftop: Optional[geopandas.geodataframe.GeoDataFrame] = None*) → geopandas.geodataframe.GeoDataFrame

Adjusts tilts which are determined to be unreasonable to standard values

Parameters **raw_PV_installations_on_rooftop** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which must contain the column “Tilt”

Returns Input GeoDataFrame where values in the “Tilt” are adjusted to standard values if they are deemed to be incorrect

Return type GeoPandas.GeoDataFrame

create_address_registry ( )
Create an address-level PV registry by grouping identified and segmented PV panels by their address.

create_rooftop_registry ( )
Create a rooftop-level PV registry by grouping identified and segmented PV panels by their rooftop id

enrich_raw_overhanging_pv_installations_with_closest_rooftop_attributes (*raw_overhanging_PV_installations: Optional[geopandas.geodataframe.GeoDataFrame] = None*, *raw_PV_installations_on_rooftop: Optional[geopandas.geodataframe.GeoDataFrame] = None*) → geopandas.geodataframe.GeoDataFrame
PV polygons which do not intersect with a rooftop polygon, although they do border to a rooftop, are matched to their nearest rooftop geometry

Parameters
- **raw_overhanging_PV_installations** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which specifies all the PV polygons which border to a rooftop, but are not intersected with a rooftop geometry
- **raw_PV_installations_on_rooftop** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which specifies all the PV polygons which are intersected with a rooftop geometry

Returns GeoDataFrame where overhanging PV installations have been enriched with the attributes of the closest rooftop

Return type GeoPandas.GeoDataFrame

filter_raw_overhanging_PV_installations_by_area (*raw_overhanging_PV_installations: Optional[geopandas.geodataframe.GeoDataFrame] = None*, *raw_PV_installations_on_rooftop: Optional[geopandas.geodataframe.GeoDataFrame] = None*) → geopandas.geodataframe.GeoDataFrame
Only overhanging PV polygons which are larger than 1 sqm in area will be kept.

Parameters
- **raw_overhanging_PV_installations** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which specifies all the PV polygons which border to a rooftop geometry.
- **raw_PV_installations_on_rooftop** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which specifies all the PV polygons which are intersected with a rooftop geometry

Returns GeoDataFrame where overhanging PV installations have been filtered to contain only those which border to a rooftop and are larger than 1 sqm in area

Return type GeoPandas.GeoDataFrame

identify_raw_overhanging_PV_installations (*raw_PV_installations_off_rooftop: Optional[geopandas.geodataframe.GeoDataFrame] = None*) → geopandas.geodataframe.GeoDataFrame
Remove PV systems from raw_PV_installations_off_rooftop which are free-standing, i.e. only use the ones belonging, i.e. bordering, to a rooftop.

Parameters **raw_PV_installations_off_rooftop** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which specifies all the PV polygons which do not intersect with a rooftop geometry. This includes free-standing PV systems and PV polygons which border to a rooftop, but couldn't be matched to that rooftop geometry initially.

Returns GeoDataFrame which specifies all the PV polygons which border to a rooftop geometry but are not yet matched with the attributes of the correspoding

rooftop. All free-standing PV system units, i.e. those which are not mounted on a rooftop, have been removed.

Return type GeoPandas.GeoDataFrame

overlay_raw_PV_installations_and_rooftops (*raw_PV_installations_gdf: Optional[geopandas.geodataframe.GeoDataFrame] = None*, *rooftop_gdf: Optional[geopandas.geodataframe.GeoDataFrame] = None*) → List[geopandas.geodataframe.GeoDataFrame]

Overlay PV polygon geometries with rooftop geometries.

Parameters
- **raw_PV_installations_gdf** (*GeoPandas.GeoDataFrame*) – GeoDataFrame with dissolved PV polygon geometries.
- **rooftop_gdf** (*GeoPandas.GeoDataFrame*) – GeoDataFrame specifying all rooftop geometries in a given county.

Returns The first element specifies PV polygons intersected with rooftop geometries, while the second element specifies PV polygons which do not intersect with rooftop geometries

Return type List[GeoPandas.GeoDataFrame]

preprocess_raw_pv_polygons (*raw_PV_polygons_gdf: geopandas.geodataframe.GeoDataFrame*, *rooftop_gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Preprocessing the raw PV polygons detected during the previous pipeline step.

Parameters
- **raw_PV_polygons_gdf** (*GeoPandas.GeoDataFrame*) – GeoPandas.GeoDataFrame consisting of all detected PV polygons from the previous pipeline step.
- **rooftop_gdf** (*GeoPandas.GeoDataFrame*) – GeoPandas.GeoDataFrame specifying all rooftop geometries and attributes within the given county

Returns GeoPandas.GeoDataFrame specifying all rooftop intersected PV polygons within a given county together with the corresponding rooftop attributes and the PV system area corrected by the rooftop's tilt.

Return type GeoPandas.GeoDataFrame

remove_erroneous_pv_polygons (*raw_PV_installations_on_rooftop: Optional[geopandas.geodataframe.GeoDataFrame] = None*) → geopandas.geodataframe.GeoDataFrame

Removes PV polygons whose aggregated intersected area is larger than their original raw area

Parameters **raw_PV_installations_on_rooftop** (*GeoPandas.GeoDataFrame*) – GeoDataFrame which must contain the columns "area_inter", "raw_area", and "identifier"

Returns Input GeoDataFrame where erroneous PV polygons have been removed

Return type GeoPandas.GeoDataFrame

Supplementary Information

PV4GER/data/nrw_county_data/
Directory which contains a GeoJSON that specifies the administrative boundaries for all counties within North Rhine-Westphalia (NRW).

PV4GER/data/nrw_rooftop_data/
Directory which contains one GeoJSON per county. The GeoJSON specifies all the rooftop information for your selected county, e.g. rooftop orientations, tilts, and geo-referenced polygons. **You need to download the respective .GeoJSON for your chosen county from our public S3 bucket as described in the README.md**.

PV4GER/data/pv_database/
Directory which contains a .csv for each analyzed county, specifying all detected PV panels by their tile ID (minx, miny, maxx, maxy coordinates), their image ID (upper left corner), and their actual geo-referenced polygon terms of latitude and longitude.

PV4GER/data/pv_registry/
Directory which contains the actual PV registry in .GeoJSON format for each analyzed county.

S

S

T

U