BUILDING FOOTPRINT EXTRACTION

DOCUMENTATION FOR “app.py”

1. Overview

This Streamlit application allows users to upload, process, and georeference raster images to extract building footprints. The application follows a multi-step workflow, including uploading raster files, tiling the images, processing tiles to extract building footprints, georeferencing the results, and finally downloading the processed data.
NOTE: The project is done in a functional programming approach, The utility functions are saved in “utilities.py”

FOR INSTALLION AND SETTING UP ENVIRONMENT, CHECKOUT “install.md”

2. Importing Libraries and Packages

- **streamlit as st**: For creating the web application interface.
- **rasterio**: For reading and processing raster files.
- **numpy as np**: For numerical operations on arrays.
- **os**: For interacting with the operating system.
- **math**: For mathematical operations.
- **Path from pathlib**: For path manipulation.
- **Image from PIL**: For handling image files.
- **show and ColorInterp from rasterio.plot**: For plotting and interpreting raster data.
- **cv2**: For computer vision operations.
- **Polygon from shapely.geometry**: For geometric operations.
- **geopandas as gpd**: For handling spatial data with Pandas.
- **utilities**: For custom utility functions (assumed to be user-defined).
- **pyproj**: For handling coordinate transformations.

3. Define Paths and Directories

- **BASE_DIR**: Directory for storing data ("./data"). Make sure it is already created.
- **MODEL_CFG_PATH**: Path to the model configuration file ("custom_cfg.yaml").
- **MODEL_WEIGHTS_PATH**: Path to the model weights file ("model_final.pth").

4. Main Function

The main() function sets up the Streamlit UI and controls the application's workflow through different pages.

4.1 Upload Raster

- **Purpose**: To upload and handle raster files.
- **Functionality**:
  - Users can upload TIFF or TIF files.
  - A unique name for the file can be provided.
  - The uploaded file is saved in a working directory.
  - Metadata of the raster file is displayed (number of bands, width, height, data type, CRS, and affine transform).
  - The TIFF file is converted to PNG for further processing.

4.2 Tiling

- **Purpose**: To divide the uploaded image into smaller tiles.
- **Functionality**:
  - Users select the number of tiles (preferably a square number).
  - The application calculates the size of each tile and creates a directory to store the tiles.
  - Tiles are created from the PNG image based on the selected configuration.

4.3 Processing

- **Purpose**: To process each tile and extract building footprints.
- **Functionality**:
  - Users select the processing device (CPU or GPU).
  - A model predictor is initialized with the specified device.
  - Each tile is processed to extract building footprint coordinates, which are saved in CSV files.
  - Progress and results are displayed to the user.

4.4 Georeferencing

- **Purpose**: To align the extracted coordinates with the geographic location of the original raster file.
- **Functionality**:
    - Georeferencing is applied to ensure that the extracted coordinates match the original image's geographic location.
    - Coordinates are adjusted for large images if necessary.
    - Errors encountered during georeferencing are reported.

4.5 Download

- **Purpose**: To provide users with the final processed output files.
- **Functionality**:
    - A combined CSV file of all extracted coordinates is created.
    - Coordinates are converted to WKT (Well-Known Text) format for easier use.
    - Users can download the combined CSV and WKT files.

5. Main Check

- Ensures that the main() function runs when the script is executed directly, initializing the Streamlit application.

DOCUMENTATION FOR "utilities.py"

1. Overview

This code is designed to process raster images for building footprint extraction using the Detectron2 model. It includes functionalities for creating directories, converting images, tiling images, and extracting building footprints from image tiles. The extracted data is then saved in a CSV format and can be converted into shapefiles for further analysis.

2. Importing Libraries and Packages

- **os**: Provides a way to interact with the operating system, such as creating directories and handling file paths.
- **csv**: Used for reading and writing CSV files.
- **cv2**: OpenCV library for image processing tasks.
- **math**: Provides mathematical functions for calculations.
- **pyproj**: For handling coordinate transformations.
- **streamlit as st**: For building the web application interface (if used in conjunction with Streamlit).
- **rasterio**: For reading and processing raster files.
- **numpy as np**: For numerical operations on arrays.
- **pandas as pd**: For data manipulation and analysis using DataFrames.
- **Image from PIL**: For handling image files.
- **Path from pathlib**: For managing file paths in a platform-independent way.
- **show and ColorInterp from rasterio.plot**: For visualizing and interpreting raster data.
- **Polygon from shapely.geometry**: For creating and manipulating geometric shapes.
- **geopandas as gpd**: For handling spatial data in DataFrames and converting to shapefiles.
- **detectron2**: For object detection using the Detectron2 model.
- **get_cfg and DefaultPredictor from detectron2.config and detectron2.engine**: For model configuration and prediction.

3. Functions

3.1 create_working_dir(unique_name, base_dir)

Creates a working directory with a unique name under a specified base directory. This is used to organize output files related to a specific project or task.

3.2 save_tif_as_png(file_path, png_path)

Converts a TIFF image to PNG format and saves it to the specified path. This function is useful for handling raster images in a format suitable for further processing.

3.3 calculate_square_tile_size(image_width, image_height, num_tiles)

Calculates the size of square tiles required to split an image into a given number of tiles. It ensures that the tiles are as uniform in size as possible.

3.4 create_tiles(image_path, tile_size, tile_dir)

Splits an image into smaller tiles of a specified size and saves them in a given directory. This is useful for processing large images in manageable pieces.

3.5 initialize_predictor(cfg_path, model_weights, device="cpu")
Initializes a Detectron2 predictor with the specified configuration file and model weights. The predictor can be set to use either a CPU or GPU for inference.

3.6 extract_coordinates_from_tile(tile_path, predictor, epsilon=6.0, output_dir='coordinates')
Processes an image tile using the Detectron2 predictor to extract building footprint coordinates. The coordinates are saved in a CSV file. This function handles the conversion of masks to polygons and saves the results.

3.7 mask_to_polygons(mask, epsilon=6.0)
Converts a binary mask to a list of polygons using contour approximation. This function is essential for turning detected objects into geometrical representations.

3.8 adjust_coordinates_for_large_image(csv_files)
Adjusts coordinates in CSV files to account for tile offsets. This ensures that the coordinates are correctly aligned with the original large image.

3.9 convert_coordinates_to_shapefiles(csv_files, output_shapefile)
Converts CSV files containing building coordinates into a single shapefile. This allows for easier integration with GIS systems and spatial analysis tools.

4. Workflow

1. **Create Working Directory**: Set up a directory for storing intermediate and output files.
2. **Convert TIFF to PNG**: Make raster images compatible for processing.
3. **Tile Image**: Split large images into smaller tiles for manageable processing.
4. **Initialize Predictor**: Set up the Detectron2 model for object detection.
5. **Extract Coordinates**: Process each tile to extract building footprints and save them.
6. **Adjust Coordinates**: Ensure extracted coordinates match the original image.
7. **Convert to Shapefiles**: Compile extracted data into a format suitable for GIS applications.