Advanced Python Programming for Remote Sensing

Task 1 – Remote Sensing Data

0. User Guide for Reproducing the Development Environment

Prerequisites

- Python 3.12.7
- Poetry

Setup of Virtual Environment (WSL Ubuntu)

1. Clone the repository:

   ```
   git clone <your-repo-url>
   cd <your-repo-fodler>
   ```

2. Install the environment using Poetry:

   ```
   poetry install --no-root
   ```

Using Poetry Shell (WSL Ubuntu)

1. Activate the virtual environment:

   ```
   $ poetry shell
   ```

2. Run `main.py` in Poetry shell:

   ```
   python main.py
   ```

3. Exit the virtual environment:

   ```
   $ exit
   ```

3. Explain why the Parquet file format is particularly well-suited for such analytical tasks. Compare it with the CSV file format and explain why, strictly speaking, the CSV file format cannot encode the same data without transforming the original data or providing additional encoding information.

Parquet Format

- Columnar Storage: Parquet is a columnar storage format, meaning data is stored column-by-column rather than row-by-row. This makes it particularly efficient for analytical tasks that require operations on specific columns, as only the necessary data is read into memory.
- Compression and Encoding: Parquet supports advanced compression techniques like Snappy and run-length encoding, reducing file size significantly and speeding up data retrieval. The format is optimized for both storage efficiency and query performance.
- Schema Enforcement: Parquet files inherently store metadata, such as data types and structures, which ensures schema enforcement. This helps avoid errors related to inconsistent or missing data structures.
- Support for Nested Data: Parquet can natively handle nested or hierarchical data structures without additional transformations, making it ideal for datasets with complex relationships or multi-dimensional data.
- Efficient Metadata Access: Parquet provides metadata indexing, which enables rapid filtering and querying without loading the entire file. This is particularly useful for large datasets like the one used in this task.

CSV Format

- Row-Based Storage: CSV is a row-based storage format, meaning all fields in a row are stored together. This makes CSV simple to use and widely supported but less efficient for columnar operations common in analytical workflows.
- No Built-In Compression: CSV does not natively support compression, leading to larger file sizes, which can be problematic for large datasets.
- Lack of Schema Information: CSV files do not store schema information, such as data types or structure. This means additional encoding (e.g., adding headers or separate metadata files) is required to interpret the data correctly, especially for hierarchical or complex datasets.
- Handling of Nested Data: CSV does not natively support nested data structures. Encoding such data into CSV often involves flattening or serialization, which can introduce complexity and data redundancy.
- Limited Metadata Support: CSV files lack built-in support for metadata. To include metadata, a separate file or manual process is required, which can be error-prone.

Why Parquet is Better-Suited for This Task

- Efficient Data Access: The task involves filtering and aggregating metadata (e.g., seasons, number of labels). Parquet's columnar format allows selective reading of relevant columns, making it significantly faster than CSV, which requires reading entire rows.
- Handling of Complex Data: The metadata includes label information, potentially stored in nested or hierarchical structures. Parquet handles this natively, whereas CSV would require flattening, leading to loss of structure or additional complexity.
- Metadata Storage: The schema and metadata stored within Parquet files simplify data interpretation, ensuring consistent and accurate processing without external metadata files or manual interventions.
- Compression: Parquet's built-in compression reduces storage space and accelerates data retrieval, which is crucial for large datasets like remote sensing metadata.

Why CSV Cannot Encode the Same Data Without Transformation

- Flat Structure: CSV assumes a flat, tabular structure, which cannot natively represent nested or hierarchical data. Nested data in the metadata would need to be serialized (e.g., JSON within a CSV cell) or flattened, which adds complexity and redundancy.
- No Type Information: CSV files do not retain information about data types or nullability. Any type enforcement requires manual intervention or external schemas, leading to potential inconsistencies.
- No Built-In Support for Metadata: Important metadata, such as the number of labels or patch structure, cannot be encoded within a CSV without additional files or transformation.

Conclusion

The Parquet format is more efficient, scalable, and suitable for analytical tasks due to its columnar storage, metadata support, and ability to handle complex data structures. While CSV remains a widely supported format for simple, flat datasets, it lacks the features required for efficient processing of structured or hierarchical datasets like remote sensing metadata.

4. Provide a brief description of how you verify the geographical information after exporting the files.

Verification of Geographical Information After Exporting

After re-tiling the GeoTIFF file into four sub-patches, it is critical to ensure that the geographical metadata, such as the Coordinate Reference System (CRS) and affine transformation, is accurately preserved or updated in the exported files. This step guarantees that the sub-patches remain geographically meaningful and can be used in further remote sensing analyses.

Verification Steps

1. Coordinate Reference System (CRS) Consistency:

- The CRS specifies how the pixel coordinates in the image relate to real-world locations. It is essential that the CRS of each exported sub-patch matches the CRS of the original GeoTIFF file.
- Verification is performed by comparing the crs attribute of the original and exported sub-patches.
- Code:

```
assert dst.crs == src.crs, f"CRS mismatch: {dst.crs} vs {src.crs}"
```

2. Affine Transformation Validation:

- The affine transformation defines how pixel coordinates are mapped to geographic coordinates. For the sub-patch _A.tif, the affine transformation should match that of the original image since it covers the same area as the top-left portion of the original file.
- For the other sub-patches (_B.tif, _C.tif, _D.tif), the affine transformation must differ, reflecting their respective geographical positions within the original image.
- Code:

```
if "_A.tif" in dst.name:
    assert dst.transform == src.transform, f"Transform should be the same: {dst.transform} vs {src.transform}"
else:
    assert dst.transform != src.transform, f"Transform should be different: {dst.transform} vs {src.transform}"
```

Importance of Validation

- Ensuring that the CRS is consistent across the original and exported files confirms that the sub-patches use the same spatial reference, avoiding potential misalignment or errors in further geospatial analyses.
- Validating the affine transformation ensures that each sub-patch correctly represents its portion of the original image in geographical space, preserving the integrity of the tiling process.

5. There are two reasons why the patches may overlap. Reflecting on the previous exercises should help you to identify one reason. Include the answer in your report and attempt to explain what the other reason could be. Note that providing specific examples is not required.

Reason 1: Re-Tiling Process
From previous exercises, it is evident that the re-tiling of large remote sensing images into smaller patches can lead to overlapping regions. During the re-tiling process:

- Adjacent patches may overlap due to fractional pixel dimensions or buffer zones used to avoid losing data along the edges of tiles.
- This is a deliberate design to ensure continuity in the data for analysis, but it results in overlaps.

Reason 2: Misalignment or Buffer Regions
Another reason for overlapping patches could be:

- Misalignment in Patch Boundaries: If the patch boundaries are not aligned correctly during the splitting process, geometries may overlap unintentionally.
- Inclusion of Buffer Regions: Buffer zones may be added around patches to account for spatial context, causing intentional overlaps for better geospatial analysis.

6.Explain the rationale behind your chosen split strategy.

The goal of this task was to create a train/test split strategy for deep learning using metadata and image patches from the dataset. The split should be non-random, reproducible, and suitable for deep learning applications.

Rationale Behind the Split Strategy:

- Label Balance: Ensure that the distribution of labels in the train and test sets is as even as possible.
- Avoid Overlap: No patch should belong to both train and test sets, ensuring that the model is evaluated on entirely unseen data.
- Reproducibility: Patches are shuffled deterministically using a fixed random seed.
- Diverse Coverage: By iterating over all unique labels and splitting their associated patches, the approach promotes diverse representation in both splits.

Key Differences from Random Sampling:

- Controlled Label Distribution: Unlike random sampling, this approach explicitly controls the label distribution in each split, ensuring balanced representation.
- Avoidance of Overlaps: The strategy ensures that no patch is mistakenly included in both sets, a problem that can occur with naive random sampling.
- Reproducibility: By using a fixed random seed, the splits can be regenerated consistently across different runs, a critical requirement for reproducibility in deep learning experiments.