Description of the Data Processing and Training Pipeline

Introduction

This document outlines, in detail, how each script in the data-processing and model-training pipeline operates. The overall flow is:

1. **`load_data_subsampled.py`** (Generates monthly data & labels in .pt files)
2. **`month_stacked_label.py`** (Stitches daily features into a single monthly feature tensor)
3. **`cov_label.py`** (Transforms monthly feature tensors into local covariance tensors for SPDNet)
4. **`basic_spdnet_pipeline.py`** (Trains/evaluates SPDNet on monthly covariance data)
5. **`unet_pipeline.py`** (Trains/evaluates U-Net on monthly stacked pixel data)
6. **`top_classes.py` & `create_clients.py`** (Groups tiles by land cover and organizes them into "client" folders)
7. **`spdnet_clientwise.py`** (Trains SPDNet on a single client folder)
8. **`unet_clientwise.py`** (Trains U-Net on a single client folder)

Sections below explain each script in this order.

`load_data_subsampled.py`

Purpose

This script reads daily GeoTIFF imagery and corresponding labels from a `/raw_data` directory. It saves them, month by month, into a new directory called `/subsampled_data`, converting each image or label into a PyTorch `.pt` file. This forms the initial, subsampled data that later scripts use.

Key Steps

- **Directory Structure (Input)**:
  - `/raw_data/planet.X/.../PF-SR/`: daily `.tif` images for a specific tile (e.g. 2018-01-05.tif).
  - `/raw_data/labels/...`: raster and vector label data organized under "Labels/Raster/" or "Labels/Vector/".
- **Subsampling and Grouping**:
  - The script is configured with `SUBSAMPLE=8`, meaning each GeoTIFF is read and reduced to 1/8th of its original spatial resolution (128×128 if the original was 1024×1024).
  - It groups daily imagery by month (e.g. `2018-01`), determined by parsing the filenames (`YYYY-MM-DD.tif`).
- **Output Files (per month)**:
  - Under `/subsampled_data/planet.X/<tile_id>/<YYYY-MM>`, the script saves:
    - `data_YYYY-MM-DD.pt`: daily image in shape $[C, H, W]$. Typically $C = 4$ if there are four image bands.
    - `labels/raster.pt`: a subsampled label raster for the same month (if available).
    - `labels/vector/...` : optional vector labels in `.pt` format.
- **Major Functionality**:
  - `process_time_series(...)`: loads daily `.tif` files, subsamples them, and saves them as `.pt` (one per day).
  - `raster_label_out_file` and `vector_label_out_file`: creation of label `.pt` files for each month.

month_stacked_label.py

Purpose

After load_data_subsampled.py produces a set of per-day data_YYYY-MM-DD.pt files, month_stacked_label.py "stacks" them into a single monthly tensor. It also processes label data (multi-class [C,H,W]) into a 2D label map [H,W] via thresholding/argmax.

Key Steps

- **Input Directory Structure**:
  - Expects a folder structure like: /subsampled_data/planet.X/<tile_id>/<YYYY-MM>/
  - Inside each month: multiple data_YYYY-MM-DD.pt plus labels/raster.pt
- **Stacking Daily Features**:
  - Each daily file is shape $[4, H, W]$.
  - If there are N days, script concatenates them along the channels dimension, resulting in shape $[H,\ W,\ 4 \times N]$.
- **Label Conversion**:
  - Label raster $[C, H, W]$ is thresholded (>127) and then argmaxed over the class dimension to get a 2D integer map $[H, W]$.
- **Output**:
  - A single pixel_dataset_<YYYY-MM>.pt is saved under

    /subsampled_data/datasets/unet/planet.X/<tile_id>/<YYYY-MM>/

    containing the following dictionary with tensor shapes

    ```
    {"features": [H,W,4*N] , "labels": [H,W]}
    ```

cov_label.py

Purpose

In cov_label.py, we convert monthly feature tensors (shape $[H, W, C]$) into per-pixel covariance matrices for use in an SPDNet model. Below is a more detailed description of how this local covariance is computed, including the formulas and tensor shapes involved.

Key Steps

1. **Load monthly data:**
   - The script reads a file named pixel_dataset_<YYYY-MM>.pt, which contains:
     - features: shape $[H,\ W,\ C]$
     - labels: shape $[H,\ W]$

     Typically, $C = 4 \times N$ (4 channels per day, N total days), or some variant depending on the data.
2. **Convert features to PyTorch format:**
   - We often permute features to $[1, C, H, W]$ so we can apply PyTorch's unfold operation for local windows:

     $$\texttt{features} \rightarrow \text{shape}\ [\ 1,\ C,\ H,\ W\ ].$$

   - We reflect-pad the outer edges by PAD, where PAD $= \left\lfloor \frac{\texttt{WINDOW\_SIZE}}{2} \right\rfloor$, ensuring each pixel's local neighborhood is well-defined.
3. **Extract local windows:**

- We use a fixed window size, e.g. `WINDOW_SIZE` = 11. PyTorch's `unfold` gives us, for each spatial position, a sub-tensor of shape $[C, \texttt{WINDOW\_SIZE}, \texttt{WINDOW\_SIZE}]$.

$$\mathbf{X} \in \mathbb{R}^{C\times(\texttt{WINDOW\_SIZE})\times(\texttt{WINDOW\_SIZE})} \to \mathbf{X} \in \mathbb{R}^{N\times C}, \text{with } N = \texttt{WINDOW\_SIZE}^2$$

4. **Center and compute covariance:**
   - We first subtract the mean over the samples to center the features:

$$\mathbf{X}_\mathrm{c} \;=\; \mathbf{X} \;-\; \frac{1}{N}\sum_{i=1}^{N}\mathbf{X}_{i,:}.$$

   - Then the empirical covariance matrix $\mathbf{\Sigma}$ is:

$$\mathbf{\Sigma} \;=\; \frac{1}{N-1}\,\mathbf{X}_\mathrm{c}^{\top}\,\mathbf{X}_\mathrm{c},$$

   whose shape is $[C,\, C]$. This is done *per pixel* in the image, thus producing $\big(H - 2\,\texttt{PAD}\big) \times \big(W - 2\,\texttt{PAD}\big)$ covariance matrices.
   - Finally, a small diagonal regularization $\alpha\,\mathbf{I}$ is added (e.g. $\alpha = 10^{-4}$) to ensure positivity. The script also calls a function `make_spd` to clamp eigenvalues above a threshold ϵ, keeping the matrix SPD (Symmetric Positive Definite).

5. **Trimming edges:**
   - Because `PAD` pixels around each edge cannot have a full `WINDOW_SIZE` × `WINDOW_SIZE` neighborhood, they are removed to match the shape of the final labels. That is, the output shape becomes:

$$\text{covariance} : [H - 2\cdot\texttt{PAD},\; W - 2\cdot\texttt{PAD},\; C,\; C], \quad \text{labels} : [H - 2\cdot\texttt{PAD},\; W - 2\cdot\texttt{PAD}].$$

6. **Output Structure:**
   - Each month's `cov_label_<YYYY-MM>.pt` is saved with:

```
{
  "covariance": [H - 2*PAD, W - 2*PAD, C, C],
  "labels":     [H - 2*PAD, W - 2*PAD]
}
```

   The top-level keys are `"covariance"` and `"labels"`.

reorg_train_val_test.py

This script reorganizes the monthly stacked images `pixel_dataset_YYYY-MM.pt` from `month_stacked_labels.py` and covariance files (`cov_label_YYYY-MM.pt` from `cov_label.py`) into three separate subdirectories: `train/`, `val/`, and `test/`. However, unlike a random or ratio-based split, it relies on lists of tile IDs specified by external text files (`train.txt`, `val.txt`, and `test.txt`) that have been downloaded or provided beforehand.

- It scans each `/subsampled_data/datasets/<spdnet_monthly,unet>/planet.*` folder for the `pixel_dataset_YYYY-MM.pt` and `cov_label_YYYY-MM.pt` files produced by `month_stacked_labels.py` and `cov_label.py`.
- For each tile ID (parsed from the `train.txt`, `val.txt`, or `test.txt` files), it identifies the corresponding tile directory and monthly subfolders, then physically moves or copies the relevant `pixel_dataset_YYYY-MM.pt` and `cov_label_*.pt` files into:

```
/subsampled_data/datasets/<spdnet_monthly,unet>/train/<tile_id>/
/subsampled_data/datasets/<spdnet_monthly,unet>/val/<tile_id>/
/subsampled_data/datasets/<spdnet_monthly,unet>/test/<tile_id>/
```

- Once done, it deletes the former directory structure.
- This ensures deep learning pipelines can later load data by referencing the appropriate subdirectory for each split.

Hence, these folders `train/`, `val/`, and `test/` now cleanly separate data based on the explicit tile IDs found in the three text files, setting the stage for network training and evaluation.

`basic_spdnet_pipeline.py`

Purpose

Trains and evaluates an SPDNet model (3-block, e.g. "SPDNet3BiRe") on the monthly covariance data created by `cov_label.py`.

Key Points

- **Input Directory**: `/subsampled_data/datasets/spdnet_monthly/<train/val/test>/`
  - Expects many `cov_label_*.pt` files, each containing `covariance` $[H \times W, C, C]$ and `labels` $[H \times W]$.
- **Data Loading**:
  - `MonthlyCovarianceDataset` flattens each pixel's $[C, C]$ matrix and label, grouping them into a dataset.
  - Typically batch size = 1, each batch may contain a set of $[N_{pixels}, C, C]$.
- **SPDNet Model**:
  - `SPDNet3BiRe` uses repeated `BiMap` → `ReEig` (and optional BatchNormSPD).
  - Finally, `LogEig` + `Vech` → a fully connected layer.
  - Key hyperparameters: `lr=1e-4`, `weight_decay=1e-4`, `epochs=...`, `batch_size=1`.
  - Uses `RiemannianAdam` for optimization.
- **Metrics**:
  - Per-class `IoU`, confusion matrix, classification report, multi-class AUC attempt, etc.

`unet_pipeline.py`

Purpose

Trains and evaluates a U-Net model on the monthly "pixel dataset" created by `month_stacked_label.py`. The data are $[H, W, 4 \times N]$ images with 2D labels $[H, W]$.

Key Points

- **Directory Structure (Input)**:
  - `/subsampled_data/datasets/unet/<train/val/test>/`
  - Each month's data: `pixel_dataset_<YYYY-MM>.pt` containing `features` → $[H, W, C]$ and `labels` → $[H, W]$.
- **UNet Model**:
  - `in_channels` $= 4 \times T$ (e.g. 112 for $T = 28$).
  - `num_classes` $= 7$.
  - Various standard conv blocks and up-conv (transpose conv) layers.
  - `pad_and_cat` ensures shape alignment in skip connections.
- **Data Loader**:
  - `UNetCropDataset` loads each `.pt` file, permutes to $[C, H, W]$, then crops or slices channels as needed (e.g. top-left $[118, 118]$).
  - Typically `batch_size` $= 1$.
- **Training Hyperparameters**:
  - `epochs` ≈ 30,
  - `lr=1e-4`,
  - `weight_decay=1e-4`.
- **Evaluation**:
  - Per-class IoU, confusion matrix, classification report, multi-class AUC, etc.
  - Best model is saved upon highest mean IoU.

`top_classes.py` and `create_clients.py`

`top_classes.py`

- Scans raw labels (`.tif` in `/raw_data/labels`) to figure out which classes dominate a tile (e.g. farmland vs. forest).
- Uses `SUBSAMPLE=8` to read label `.tif` quickly.
- Finds the top n classes for each tile by counting pixel frequencies.
- Outputs a dictionary of tile IDs to the top classes.

`create_clients.py`

- Reads the tile grouping from `top_classes.py`.
- Copies data from `/subsampled_data/datasets/(unet or spdnet_monthly)/` into "client" directories based on each tile's land-cover group (e.g. "mixed," "urban," etc.).
- Final structure: `/clients/unet/<group>/<tile_id>` or `/clients/spdnet_monthly/<group>/<tile_id>`.

`spdnet_clientwise.py`

Purpose

Trains the same SPDNet approach (as in `basic_spdnet_pipeline.py`) but for a single "client" folder, e.g. a group of tiles with similar dominant classes.

Process

- Points to a client path, e.g. `/clients/spdnet_monthly/mixed/`.
- Loads all the `cov_label_*.pt` files in that directory.
- Splits them (70%-15%-15%) for train-val-test.
- Hyperparameters are similar: `batch_size` $= 1$, `epochs` $= 15$, `lr` $= 1e-4$.
- The model is the same `SPDNet3BiRe` with RiemannianAdam.

`unet_clientwise.py`

Purpose

Trains a U-Net on a single "client" folder containing stacked monthly pixel data.

Details

- Reads from e.g. `/clients/unet/mixed/`, loads `pixel_dataset_*.pt` files.
- Splits them 70%-15%-15%.
- Hyperparameters: `epochs` $= 15$, `lr` $= 1e-4$, `weight_decay` $= 1e-4$.
- Produces final metrics (accuracy, IoU, confusion matrix, etc.).

Conclusion

This pipeline starts from raw daily `.tif` images and ends in multiple model training strategies, both U-Net-based (for direct pixel classification) and SPDNet-based (for classification via covariance matrices). The scripts support flexible grouping of tiles into "clients," enabling clientwise or federated learning setups.