Deep Learning Global Health Analytics
Software Installation and Configuration Guide

This guide covers three essential components: software installation, file system structure, and notebook configuration settings. It details how to set up the required Python development environment on a Linux system and configure the project-specific software available on GitHub. Additionally, it provides an overview of the project's file system structure, including the notebooks used for pre-processing geospatial data and supporting model fine-tuning. While no data is stored in the repository due to its large size, the folders designated for storing specific datasets are clearly identified.

The installation and configurations described in this document have been tested on macOS and Ubuntu.

1. Development Environment

The first step in the installation process is to create a conda virtual environment using Python 3.9 with the following command. The '$' represents the terminal command prompt:

```
~$ conda create -n py39-pt-test python=3.9
```

Activate the environment as follows:

```
~$ conda activate py39-pt
```

Once activated, the command line will reflect the activated environment: `(py39-pt) ~$`

Next, install the following Python packages using conda:

```
(py39-pt) ~$ conda install pytorch torchvision -c pytorch
(py39-pt) ~$ conda install numpy pandas pyreadstat
(py39-pt) ~$ conda install geopandas pyproj rasterio shapely gdal
(py39-pt) ~$ conda install tqdm
(py39-pt) ~$ conda install matplotlib seaborn bokeh pillow
(py39-pt) ~$ conda install scikit-learn umap-learn
(py39-pt) ~$ conda install ipykernel jupyter
(py39-pt) ~$ conda install glob
(py39-pt) ~$ conda install selenium
```

Some packages are not available through `Conda` and must be installed using `pip` as follows:

```
(py39-pt) ~$ pip install torchinfo torchmetrics torchgeo scikit-gstat
```

To register the current Python environment as a Jupyter kernel, so it can be selected within the Jupyter interface, execute the following command (this is a one-time step):

```
(py39-pt) ~$ python -m ipykernel install --name=py39-pt --display-name “Python (py39-pt)”
```

Before launching the Jupyter notebook server, change directory to the top level project directory indicated below:

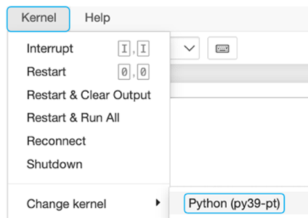
```
(py39-pt) ~/Deep-Learning-Global-Health-Analytics$
```

Next, issue the following command to launch the Jupyter Notebook interface in your default web browser. This can be executed from the command terminal from within the conda virtual environment or directly from the command terminal from outside the virtual environment.

```
(py39-pt) ~/Deep-Learning-Global-Health-Analytics$ jupyter notebook &
```

Since the kernel was registered to the virtual environment in the previous command, it can be selected from within the notebook.

After issuing the above command, you can launch any of the Jupyter notebooks in the project. However, before executing the code cells, make sure to go to the **"Kernel"** pulldown menu and select **"Change Kernel"** to confirm or set the kernel associated with the Conda virtual environment created earlier.



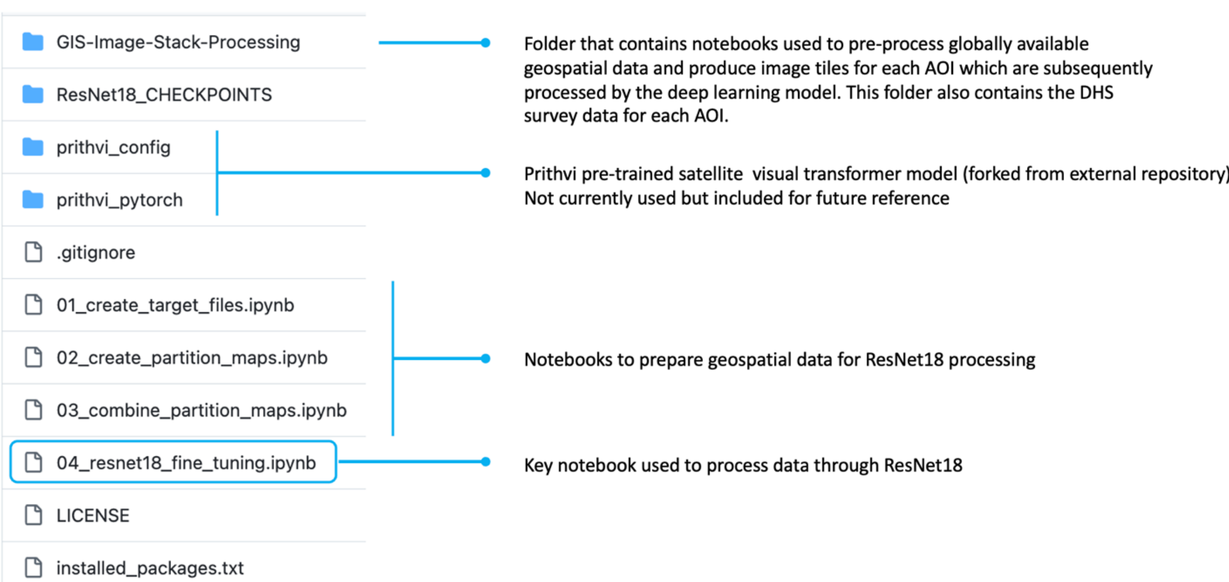
Note that the interaction with Jupyter notebooks may differ slightly depending on whether you're using **JupyterLab** or **Jupyter Notebook** (both interfaces for running Jupyter kernels). JupyterLab offers a more modern and flexible interface, but the code should execute the same in either environment.

2. Project Software

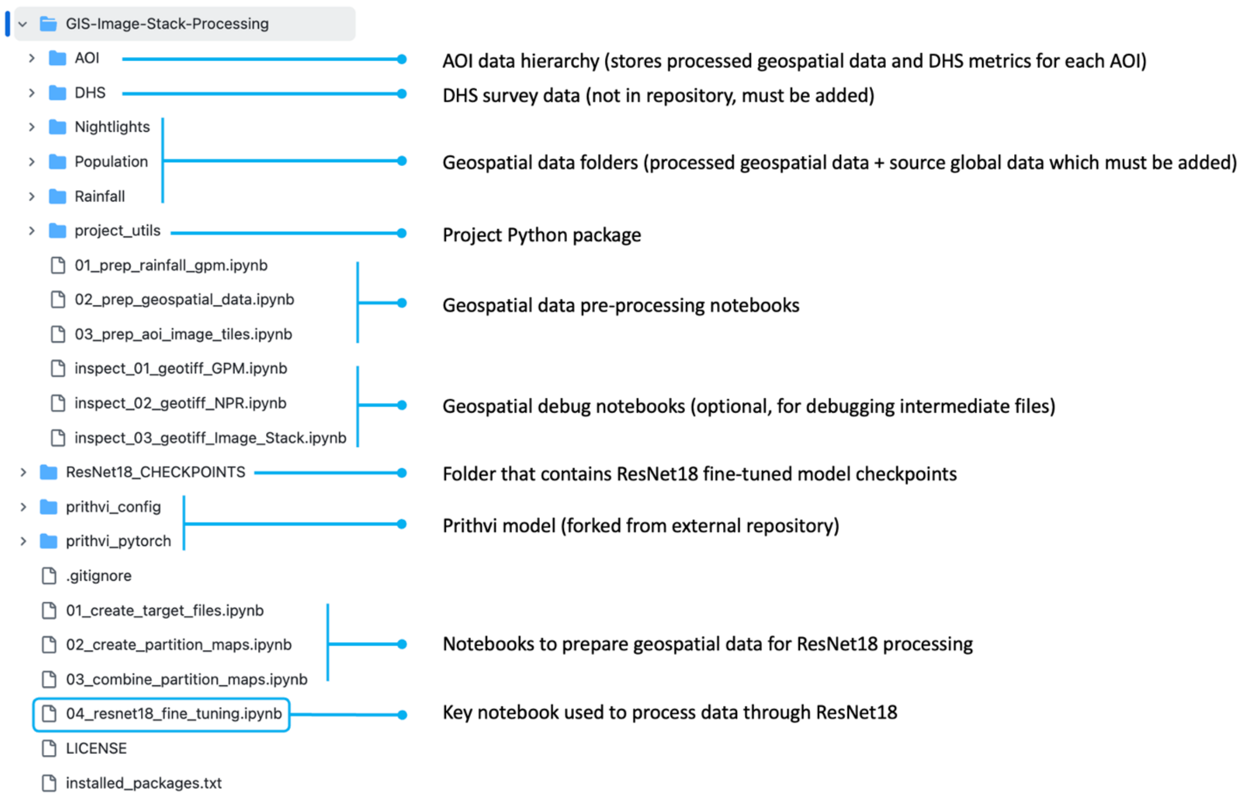
The software for this project is located at the following GitHub repository:

https://github.com/kromydas/Deep-Learning-Global-Health-Analytics/

The top-level folder hierarchy is shown below which identifies key folders and files.



The expanded view below shows the content of `GIS-Image-Stack-Processing`, which contains several notebooks used to pre-process geospatial data. This folder also contains a project Python package `project_utils` with lower-level functions that support most of the notebooks in the repository. At the top-level there are also several notebooks that are related to preparing the geospatial data for processing the through the deep learning model (ResNet18).



The project Python package `project_utils`, contains many lower-level functions. The advantage of the Python package is that it allows these functions to be defined outside of the notebooks, reducing code clutter within the notebooks. The `project_utils` package consists of the following modules:

```
project_utils/
        aoi_configurations.py
        gist_utils.py
        plot_utils.py
        resnet_utils.py
```

There is also code related to the deep learning model Prithvi-100M, a pre-trained satellite visual transformer. This code was forked from the following repository:

https://github.com/isaaccorley/prithvi-pytorch

Although this code is not currently in use, it is included in the repository for potential future reference.

2.1 GIS-Image-Stack-Processing (Notebooks)

There are two main sets of notebooks associated with this project. The first set is located in `` `GIS-Image-Stack-Processing` `` and are used to pre-process globally available geospatial data and produce image tiles for each AOI which are subsequently processed by the deep learning model. This set of notebooks therefore serve the purpose of preparing the geospatial data, but only need to be executed "once" as a set although each notebook is executed multiple times , as described below to generate the image tiles for each AOI.

Notebook	Notes
`01_prep_rainfall_gpm.ipynb`	**Pre-process global GPM rainfall data**. Converts multi-year monthly averages to a single global daily average for each AOI. This notebook should be executed once for each AOI. The output from this notebook is a daily average rainfall GeoTiff file for each AOI as shown in the example below: `./GIS-Image-Stack-Processing` `/Rainfall` `/GMP_2001-2022/` `PK/AOI_Crop_Daily/` `GPM_2001-2022.01.V07B_PK_avg.tif`
`02_prep_geospatial_data.ipynb`	**Pre-process each geospatial data type for each AOI**. Creates AOI spatially aligned image stacks at a consistent and specified resolution. This notebook should be executed once for each data type for each AOI. For example, for the PK AOI specified, it should be executed once for Nightlights, once for Population and once for Rainfall. The output files from each execution represent the three channels in the AOI image stack (i.e., three large GeoTiff files covering the entire AOI).
`03_prep_aoi_image_tiles.ipynb`	**Create spatially aligned image tiles**: This notebook uses the AOI image stack to create smaller image tiles centered at the DHS survey locations. The AOI image stack from the previous step is copied to a new location on the file system intentionally for use with this notebook. The output from this notebook are 224x224 image tiles (GeoTiff files) located within each AOI. `./AOI/PK/Image_Tiles/` `Nightlights/` `PK_1_C-1_Nightlights_2022_400m.tif` `:` `Population/` `PK_1_C-1_Population_2022_400m.tif` `:` `Rainfall/` `PK_1_C-1_Rainfall_2001 2022_400m.tif` `:`

2.2 Deep-Learning-Global-Health-Analytics (Model Preparation Notebooks)

The next set of notebooks prepare the geospatial data for ResNet18 processing. This process involves loading DHS survey data, computing cluster-level statistics for each AOI, and storing that data in JSON files for downstream processing. Plots are also generated to visualize the spatial distribution of the metrics, and variograms are produced as well. The target files for each AOI (e.g., `./AOI/PK/Targets/targets.json`) are then used as input (along with a specified criterion) to create virtual partition maps for training and validation, supporting model fine-tuning.

<table>
<tr><th>Notebook</th><th>Notes</th></tr>
<tr><td><code>01_create_target_files.ipynb</code></td><td>Create target files: This notebook loads and processes the DHS recode files for the specified AOI, computing cluster-level metrics. These metrics, along with the cluster IDs and their (lat, lon) coordinates, are stored in a <code>targets.json</code> file. Additionally, survey metrics are plotted on a geographical map to visualize their spatial distribution. Variograms for each metric are also generated. The resulting plots are saved to disk in the following directories, which are automatically created if they do not already exist.

<code>Plots_Geospatial/</code>
<code>Plots_Variograms/</code>

Since the notebook serves multiple purposes the creation of target files and geospatial maps are both configurable.</td></tr>
<tr><td><code>02_create_partition_maps.ipynb</code></td><td>Create virtual partition maps: This notebook uses DHS cluster data to virtually partition the clusters into training and validation sets. The current partitioning criterion applies a longitude threshold to minimize spatial autocorrelation between the training and validation datasets, though this can be adjusted to use other methods. The notebook generates three JSON files in the following location:

<pre>./GIS-Image-Stack Processing/
    AOI/
        Partitions/
            {country_code}/
                all.json
                train.json
                valid.json</pre>
Each file contains a JSON object where the two-letter country code is the key, and the value is an array of integers representing the cluster IDs for that partition. This structure allows for flexible, virtual data partitioning, making it easy to modify the partitioning logic as needed.</td></tr>
</table>

Notebook	Notes
`03_combine_partition_maps.ipynb`	**Combine partition maps**: This notebook combines the AOI-specific partition maps into a master set based on the provided list of AOIs. The master partition maps are located directly within the Partitions folder: `./GIS-Image-Stack-Processing` `    /AOI/` `        Partitions/` `            all.json` `            train.json` `            valid.json` These master partition files are used to create training and validation datasets for fine-tuning deep learning models. The AOI-specific partition maps generated by the previous notebook serve as a pre-processing step to enable the aggregation of the AOI-specific maps into a master set. The master `all.json` file is not used for training but represents the superset of all cluster IDs for a given AOI and serves as a reference.

2.3 Deep-Learning-Global-Health-Analytics (ResNet18 Processing Notebook)

The primary analysis notebook warrants its own section, as there are several configurations that require explanation. This notebook leverages the ResNet18 deep learning model for processing geospatial data in an unsupervised learning and cluster analysis workflow.

```
04_resnet18_fine_tuning.ipynb
```

In this notebook, for each data sample (DHS cluster), features are extracted from the ResNet18 model's backbone. After feature extraction UMAP (or PCA) are applied to reduce the dimensionality of the feature space. This greatly simplifies the high-dimensional feature space, making it more suitable for clustering algorithms. Once reduced, clustering is performed on the projected data using K-Means. Each data point is then assigned a cluster label, which is visualized on a geographical map to show the spatial distribution of clusters. By comparing these spatial distributions with known vaccination rates, we can assess the feasibility of using this approach to identify potential disease hotspots.

The notebook supports three run modes (previous listed), each following the same processing steps but differing in how the deep learning model is used.

Before discussing the details for each run mode, it’s important to specify the AOI at the top of the notebook using its two-letter country code. Regardless of the run mode, an AOI must be specified for analysis. In the example below, Pakistan is specified as the AOI, and the associated image tiles for Pakistan are processed through the ResNet18 model.

Required Configurations

The following configurations are required for each execution of this notebook: the two-letter country code. Other model and feature extraction configurations are available in the Configuraton section.

```
country_code  = 'PK'      # Set the country code to one of the available AOIs in the list below

Available AOIs: AM (Armenia)
                MA (Morocco)
                MB (Moldova)
                ML (Mali)
                MR (Mauritania)
                NI (Niger)
                PK (Pakistan)
                SN (Senegal)
                TD (Chad)
```

```
#--------------------------------------------------
# REQUIRED CONFIGURATIONS HERE
#--------------------------------------------------
country_code  = 'PK'      # Set the country code
#--------------------------------------------------
```

The notebook is highly configurable however, for the given DHS survey metrics, there are limited configurations that can be specified by the user. Beyond the analysis AOI specified above, most other configurations have acceptable defaults. Still, there are a few that can be adjusted for experimentation. The screenshot below shows the first half of the configuration settings available in the notebook. Notice that only the last section contains configurations that can be edited to specify the feature extraction layer from the ResNet18 model backbone.

```
# Enumerated list of DHS target values
class TargetType(Enum):
    FRACTION_DPT3_VACCINATED  = "fraction_dpt3_vaccinated"
    FRACTION_WITH_ELECTRICITY = "fraction_with_electricity"
    FRACTION_WITH_FRESH_WATER = "fraction_with_fresh_water"
    MEAN_WEALTH_INDEX         = "mean_wealth_index"
    FRACTION_WITH_RADIO       = "fraction_with_radio"
    FRACTION_WITH_TV          = "fraction_with_tv"


target_name_mapping = {
    TargetType.FRACTION_DPT3_VACCINATED:  "Fraction DPT3 Vaccinated",
    TargetType.FRACTION_WITH_ELECTRICITY: "Fraction with Electricity",
    TargetType.FRACTION_WITH_FRESH_WATER: "Fraction with Fresh Water",
    TargetType.MEAN_WEALTH_INDEX:         "Mean Wealth Index",
    TargetType.FRACTION_WITH_RADIO:       "Fraction with Radio",
    TargetType.FRACTION_WITH_TV:          "Fraction with TV",
}

class ModelMode(Enum):
    PRE_TRAINED = "Pre_Trained"
    FINE_TUNE   = "Fine_Tune"
    CHECKPOINT  = "Checkpoint"

# Dataset configuration parameters
@dataclass(frozen=True)
class DatasetConfig:
    COUNTRY_CODE:     str
    IMG_HEIGHT:       int = 224
    IMG_WIDTH:        int = 224
    GIS_ROOT:         str = './GIS-Image-Stack-Processing'
    AOI_ROOT:         str = './GIS-Image-Stack-Processing/AOI/'
    PRT_ROOT:         str = './GIS-Image-Stack-Processing/AOI/Partitions'
    TARGET_TYPE:      Union[TargetType, List[TargetType]] = TargetType.FRACTION_WITH_FRESH_WATER

@dataclass(frozen=True)
class FeatureConfig:
    FEATURE_LAYER:   str = 'layer4'
    BLOCK_INDEX:     int = 1
    SUB_LAYER_PART:  str = 'conv2'
    RELU:            bool = True        # Set to True to extract featuers from last (ReLU) in layer.
                                        # Ignores BLOCK_INDEX and SUB_LAYER_PART
```

These setting should NOT be edited

These settings can be edited to experiment with different extraction points

The table on the following page shows the ResNet18 model architecture summary, including all the layer names. The configuration structure above allows the user to specify the Sequential layer name (`FEATURE_LAYER`), the BasicBlock (`BLOCK_INDEX`), and the Sub-layer name (`SUB_LAYER_PART`).

There is one exception that does not fit this specification which is the ReLU layer at the end of each sequential layer. Specifically, if the `relu` sublayer is the desired extraction point, then the feature and layer must be specificed and the RELU input must be specified as True. In the example above, the `BLOCK_INDEX` and `SUB_LAYER_PART` are both ignored when `RELU` is True. This also means that all other layers (other than relu) should be specificed as described above, with `RELU = False`.

```
==========================================================================================
Layer (type (var_name))                  Output Shape              Param #
==========================================================================================
ResNet (ResNet)                          [1, 5]                    --
├─Conv2d (conv1)                         [1, 64, 112, 112]         (9,408)
├─BatchNorm2d (bn1)                      [1, 64, 112, 112]         (128)
├─ReLU (relu)                            [1, 64, 112, 112]         --
├─MaxPool2d (maxpool)                    [1, 64, 56, 56]           --
├─Sequential (layer1)                    [1, 64, 56, 56]           --
│    └─BasicBlock (0)                    [1, 64, 56, 56]           --
│    │    └─Conv2d (conv1)               [1, 64, 56, 56]           36,864
│    │    └─BatchNorm2d (bn1)            [1, 64, 56, 56]           128
│    │    └─ReLU (relu)                  [1, 64, 56, 56]           --
│    │    └─Conv2d (conv2)               [1, 64, 56, 56]           36,864
│    │    └─BatchNorm2d (bn2)            [1, 64, 56, 56]           128
│    │    └─ReLU (relu)                  [1, 64, 56, 56]           --
│    └─BasicBlock (1)                    [1, 64, 56, 56]           --
│    │    └─Conv2d (conv1)               [1, 64, 56, 56]           36,864
│    │    └─BatchNorm2d (bn1)            [1, 64, 56, 56]           128
│    │    └─ReLU (relu)                  [1, 64, 56, 56]           --
│    │    └─Conv2d (conv2)               [1, 64, 56, 56]           36,864
│    │    └─BatchNorm2d (bn2)            [1, 64, 56, 56]           128
│    │    └─ReLU (relu)                  [1, 64, 56, 56]           --
├─Sequential (layer2)                    [1, 128, 28, 28]          --
│    └─BasicBlock (0)                    [1, 128, 28, 28]          --
│    │    └─Conv2d (conv1)               [1, 128, 28, 28]          73,728
│    │    └─BatchNorm2d (bn1)            [1, 128, 28, 28]          256
│    │    └─ReLU (relu)                  [1, 128, 28, 28]          --
│    │    └─Conv2d (conv2)               [1, 128, 28, 28]          147,456
│    │    └─BatchNorm2d (bn2)            [1, 128, 28, 28]          256
│    │    └─Sequential (downsample)      [1, 128, 28, 28]          8,448
│    │    └─ReLU (relu)                  [1, 128, 28, 28]          --
│    └─BasicBlock (1)                    [1, 128, 28, 28]          --
│    │    └─Conv2d (conv1)               [1, 128, 28, 28]          147,456
│    │    └─BatchNorm2d (bn1)            [1, 128, 28, 28]          256
│    │    └─ReLU (relu)                  [1, 128, 28, 28]          --
│    │    └─Conv2d (conv2)               [1, 128, 28, 28]          147,456
│    │    └─BatchNorm2d (bn2)            [1, 128, 28, 28]          256
│    │    └─ReLU (relu)                  [1, 128, 28, 28]          --
├─Sequential (layer3)                    [1, 256, 14, 14]          --
│    └─BasicBlock (0)                    [1, 256, 14, 14]          --
│    │    └─Conv2d (conv1)               [1, 256, 14, 14]          294,912
│    │    └─BatchNorm2d (bn1)            [1, 256, 14, 14]          512
│    │    └─ReLU (relu)                  [1, 256, 14, 14]          --
│    │    └─Conv2d (conv2)               [1, 256, 14, 14]          589,824
│    │    └─BatchNorm2d (bn2)            [1, 256, 14, 14]          512
│    │    └─Sequential (downsample)      [1, 256, 14, 14]          33,280
│    │    └─ReLU (relu)                  [1, 256, 14, 14]          --
│    └─BasicBlock (1)                    [1, 256, 14, 14]          --
│    │    └─Conv2d (conv1)               [1, 256, 14, 14]          589,824
│    │    └─BatchNorm2d (bn1)            [1, 256, 14, 14]          512
│    │    └─ReLU (relu)                  [1, 256, 14, 14]          --
│    │    └─Conv2d (conv2)               [1, 256, 14, 14]          589,824
│    │    └─BatchNorm2d (bn2)            [1, 256, 14, 14]          512
│    │    └─ReLU (relu)                  [1, 256, 14, 14]          --
├─Sequential (layer4)                    [1, 512, 7, 7]            --
│    └─BasicBlock (0)                    [1, 512, 7, 7]            --
│    │    └─Conv2d (conv1)               [1, 512, 7, 7]            1,179,648
│    │    └─BatchNorm2d (bn1)            [1, 512, 7, 7]            1,024
│    │    └─ReLU (relu)                  [1, 512, 7, 7]            --
│    │    └─Conv2d (conv2)               [1, 512, 7, 7]            2,359,296
│    │    └─BatchNorm2d (bn2)            [1, 512, 7, 7]            1,024
│    │    └─Sequential (downsample)      [1, 512, 7, 7]            132,096
│    │    └─ReLU (relu)                  [1, 512, 7, 7]            --
│    └─BasicBlock (1)                    [1, 512, 7, 7]            --
│    │    └─Conv2d (conv1)               [1, 512, 7, 7]            2,359,296
│    │    └─BatchNorm2d (bn1)            [1, 512, 7, 7]            1,024
│    │    └─ReLU (relu)                  [1, 512, 7, 7]            --
│    │    └─Conv2d (conv2)               [1, 512, 7, 7]            2,359,296
│    │    └─BatchNorm2d (bn2)            [1, 512, 7, 7]            1,024
│    │    └─ReLU (relu)                  [1, 512, 7, 7]            --
├─AdaptiveAvgPool2d (avgpool)            [1, 512, 1, 1]            --
├─Sequential (fc)                        [1, 5]                    --
│    └─Dropout (0)                       [1, 512]                  --
│    └─Linear (1)                        [1, 5]                    2,565
==========================================================================================
```

The next section below contains two more sections that can be edited. The most important configurtion below is in the `TrainingConfig` block in which the user specifies the model run mode based on the enumeration described earlier.

```
class ModelMode(Enum):
    PRE_TRAINED = "Pre_Trained"
    FINE_TUNE   = "Fine_Tune"
    CHECKPOINT  = "Checkpoint"
```

The user can specify one of three options to set the run mode. The default run mode is: `ModelMode.CHECKPOINT`, but this requires a checkpoint file to be located in:

```
./ResNet18_CHECKPOINTS
    ResNet18_4layers_5targets_lr_4e-05_bs_8_l2_0.02_do_0.4_ft_4_log_1_aug_1_trg_5__9AOI.pth
```

<u>It's important to note that when this run mode is specified, the notebook will search for a specific checkpoint file name that matches the current training configuration settings</u>. These settings are used to construct the filename that the notebook will attempt to locate in the file system. While this type of configuration can be a bit tedious and could be improved by allowing the user to specify the filename directly, the current version of the notebook only supports the described approach.

If any issues are encountered while using the default run mode, it is recommended to switch the run mode to `ModelMode.PRE_TRAINED`, in which the ResNet18 pre-trained model is used directly. The pre-trained model should produce results very similar to the fine-tuned model.

```
@dataclass(frozen=True)
class TrainingConfig:
    OUTPUTS:          int
    VERBOSE:          bool  = True
    EPOCHS:           int   = 51
    LEARNING_RATE:    float = .00004
    BATCH_SIZE:       int   = 8
    L2_REG:           float = .02
    DROPOUT:          float = .4
    PATIENCE:         int   = 10
    NUM_WORKERS:      int   = num_workers
    FINE_TUNE_LAYERS: int   = 4
    USE_DATA_AUG:     bool  = True
    USE_LOG1P:        bool  = True
    MODEL_MODE:       ModelMode = ModelMode.CHECKPOINT  # Specify the type of model
    LOG_DIR:          str   = "./ResNet18_LOGS_DATA"
    CHECKPOINT_DIR:   str   = "./ResNet18_CHECKPOINTS"
    CASE_STRING:      str   = "9AOI"                    # Optional: Additional case string

    # 9 AOI Values
    MEAN_STD: dict = field(default_factory=lambda: {
        'Nightlights': (0.4051, 0.3869),
        'Population':  (1.5032, 2.2611),
        'Rainfall':    (1.4170, 0.8373)
    })

    def get_checkpoint_file(self, training_string: str = "") -> str:
        case_str = training_string if training_string else self.CASE_STRING
        return f"ResNet18_{self.FINE_TUNE_LAYERS}layers_{self.OUTPUTS}targets_{case_str}.pth"

    @staticmethod
    def from_dataset_config(dataset_config: DatasetConfig):
        if isinstance(dataset_config.TARGET_TYPE, list):
            num_targets =  len(dataset_config.TARGET_TYPE)
        else:
            num_targets = 1
        return  TrainingConfig(OUTPUTS=num_targets)


# Result configurations
@dataclass(frozen=True)
class ResultsConfig:
    COMPUTE_GEOSPATIAL:  bool = True
    COMPUTE_AOI_DIST:    bool = False  # Takes time when set to True
    PLOT_TRAINING_DIR:    str = './Plots_Fine_Tuning'
    PLOT_UMAP_FEAT_DIR:   str = './Plots_Projected_Features/9AOI'
    PLOT_GEOSPATIAL_DIR:  str = './Plots_Geospatial/9AOI'
```

Change this to specify the run mode

These setting can be edited, but are only applicable when ModelMode = ModelMode.FINE_TUNE

These setting should NOT be edited

Good defaults, but can edit to customize output

The section of the notebook shown below is an advanced configuration section, which only applies when fine-tuning the model (i.e., `ModelMode = ModelMode.FINE_TUNE`). It allows the user to configure the regression head for one or more DHS survey metrics. This section will not be discussed further at this time, as model fine-tuning is an advanced run mode that requires a GPU and experience with training deep learning models. It is mentioned here so that users are aware that this section should not be edited unless the intent is to fine-tune the model.

```python
#---------------------------------------------------------------------------------
# Multiple Regression
# dataset_config = DatasetConfig(COUNTRY_CODE=country_code,
#                                TARGET_TYPE=[TargetType.FRACTION_DPT3_VACCINATED,
#                                             TargetType.FRACTION_WITH_ELECTRICITY,
#                                             TargetType.FRACTION_WITH_FRESH_WATER,
#                                             TargetType.MEAN_WEALTH_INDEX,
#                                             TargetType.FRACTION_WITH_RADIO,
#                                             TargetType.FRACTION_WITH_TV])

dataset_config = DatasetConfig(COUNTRY_CODE=country_code,
                              TARGET_TYPE=[TargetType.FRACTION_DPT3_VACCINATED,
                                           TargetType.FRACTION_WITH_ELECTRICITY,
                                           TargetType.MEAN_WEALTH_INDEX,
                                           TargetType.FRACTION_WITH_RADIO,
                                           TargetType.FRACTION_WITH_TV])

# dataset_config = DatasetConfig(COUNTRY_CODE=country_code,
#                                TARGET_TYPE=[TargetType.FRACTION_DPT3_VACCINATED,
#                                             TargetType.FRACTION_WITH_ELECTRICITY,
#                                             TargetType.FRACTION_WITH_FRESH_WATER,
#                                             TargetType.MEAN_WEALTH_INDEX])

for target in dataset_config.TARGET_TYPE:
    print(f"Selected Target: {target_name_mapping[target]}")

desired_target_type_key = ""
#---------------------------------------------------------------------------------



#---------------------------------------------------------------------------------
# # Or, Single Regression

# # Define a mapping between target type strings and the corresponding TargetType enums
# target_type_mapping = {
#     'dpt3'        : [TargetType.FRACTION_DPT3_VACCINATED],
#     'wealth'      : [TargetType.MEAN_WEALTH_INDEX],
#     'electricity' : [TargetType.FRACTION_WITH_ELECTRICITY],
#     'water'       : [TargetType.FRACTION_WITH_FRESH_WATER]
# }

# desired_target_type_key = 'dpt3'  # Specify which metric to use for single regression

# # Use the desired target type key to select the appropriate TARGET_TYPE
# dataset_config = DatasetConfig(
#     COUNTRY_CODE=country_code,
#     TARGET_TYPE=target_type_mapping[desired_target_type_key]
# )
#---------------------------------------------------------------------------------
```

Further down in the notebook, there is a final section that contains a user-level configuration. Here, the user can select the type of projection to use (UMAP or PCA). The reason this section is included at this point in the notebook is to facilitate experimentation with different projection methods, where access to the plotted projected data is convenient.

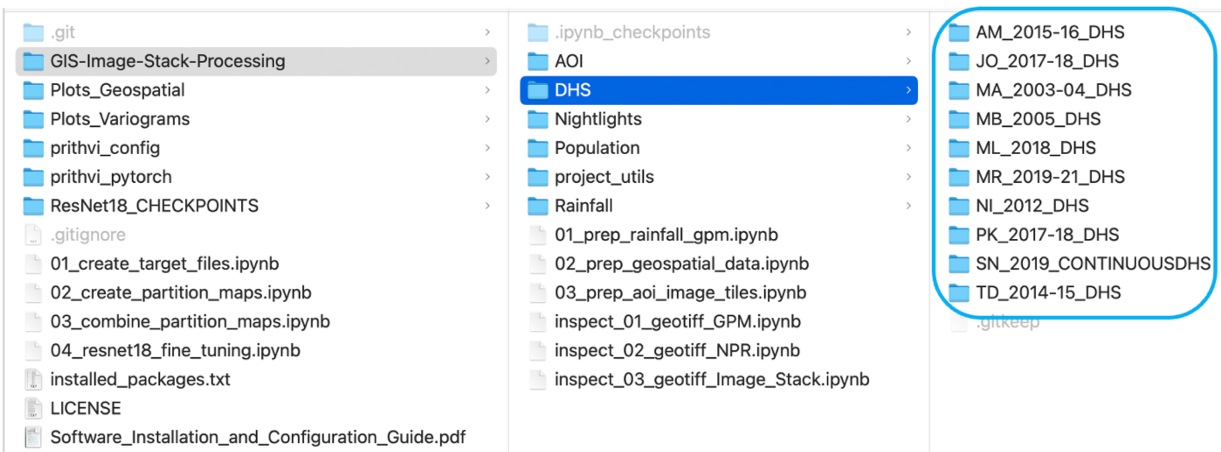
```python
class ProjectionType(Enum):
    PCA  = "PCA"
    UMAP = "UMAP"

#----------------------------------------
# *** SET PROJECTION TYPE HERE ***
#----------------------------------------
projection_type = ProjectionType.UMAP
#----------------------------------------


if projection_type == ProjectionType.UMAP: # STOP: DO NOT EDIT THIS BY MISTAKE!

    projected_features = project_umap(features_standardized,
                                      n_components=2,
                                      n_neighbors=15,
                                      min_dist=0.1)
else:
    pca = PCA(n_components=10)
    pca.fit(features_standardized)

    # Transform the data
    projected_features = pca.transform(features_standardized)
```

The projection type can be edited here

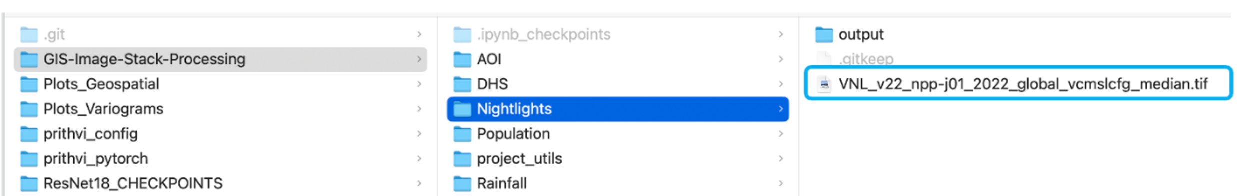
3. Project Data

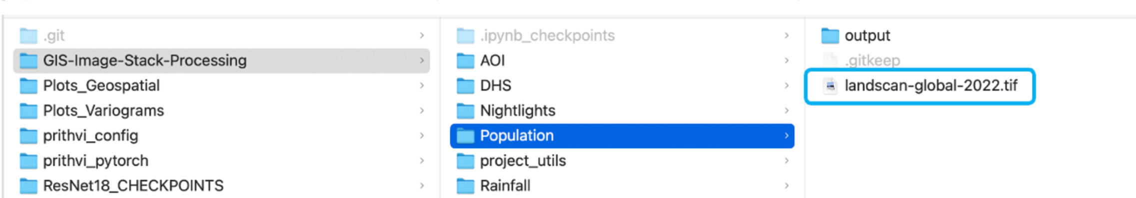
As previously discussed, several types of data need to be placed in specific folders within the distribution. The following sections below show where all the input data should be located, ensuring that all the notebooks in the repository can be executed from start to finish to pre-process the geospatial data, create AOI image tiles, and prepare the data for ResNet processing. However, since geospatial data preprocessing and image tile generation are one-time steps, it is also possible to directly populate the AOI folder with these pre-processed files and proceed from there to use the ResNet18 model notebook. Both options will be covered below.

The filesystem view below shows the location for the AOI DHS data. An archive of this data will be made available for download, after which it should be extracted to the DHS folder.

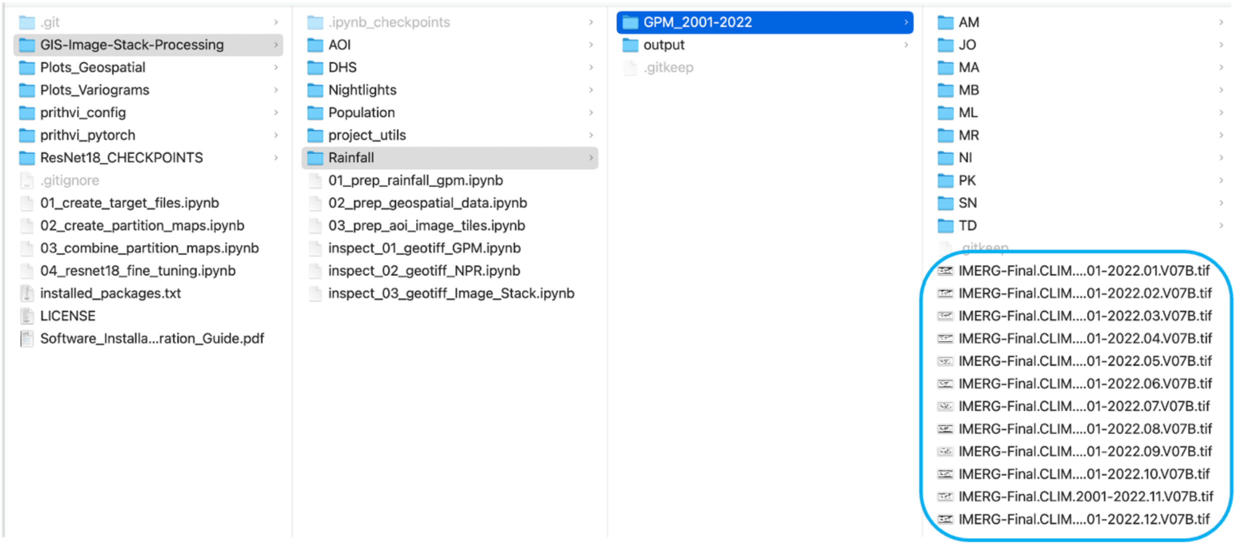


Each geospatial data type contains global source files. For Nightlights and Population, these are single (large) GeoTIFF files that span the globe. Each file should be placed in its respective folder, as shown below.

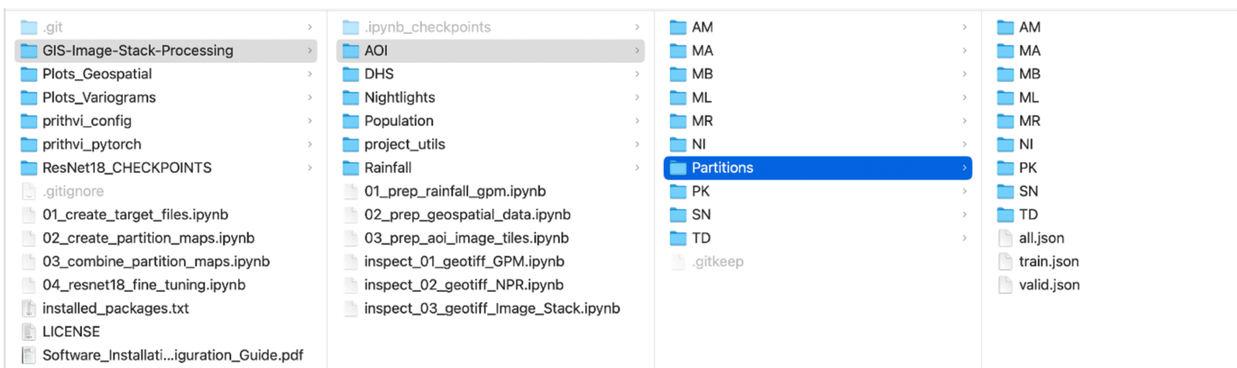


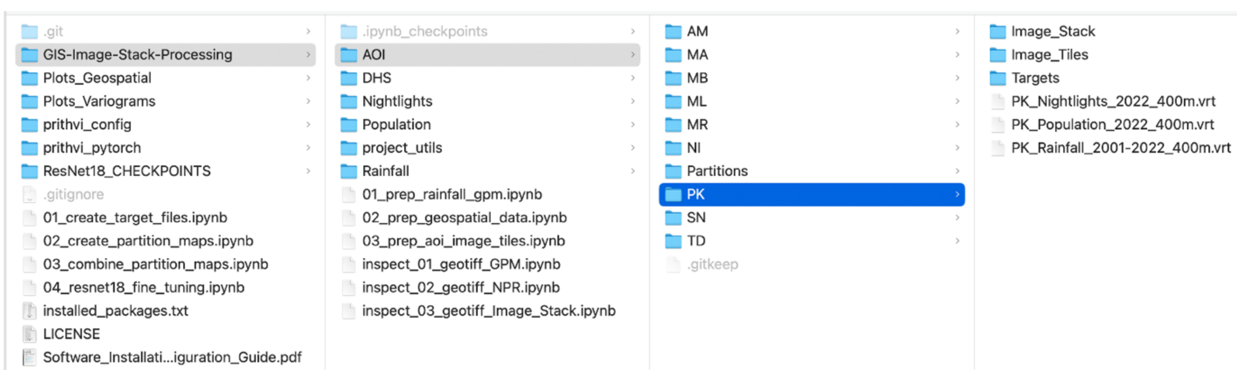


The source data for Rainfall is more complicated since it is provided in 12 monthly files. These files should be placed in the `GPM_2001-2022` folder, as shown below. The AOI folders contained in the `GPM_2001-2022` folder are produced by the first notebook (`01_prep_rainfall_gpm.ipynb`)



After all this data has been processed, the AOI folder will be populated as shown in the view below. An archive will be provided containing all this data, so it won't be necessary to process everything from scratch, which can be tedious and time-consuming.





Appendix: Notebook Configuration Notes

Three notebooks require the use if binary files as explained in the comments below. Therefore, the pathname to the bin folder in the Conda virtual environment should be edited to reflect the location of this folder on the filesystem.

```
01_prep_rainfall_gpm.ipynb
02_prep_geospatial_data.ipynb
03_prep_aoi_image_tiles.ipynb
```

```
#--------------------------------------------------------------------------------------
# *** IMPORTANT: SYSTEM PATH TO SET ***
#--------------------------------------------------------------------------------------
# The following path is required, as it contains GDAL binaries used for several
# pre-processing functions. The pathname corresponds to the Conda virtual environment
# created for this project (e.g., "py39-pt").
#
# Note: GDAL was adopted as a benchmark to compare the original GIS data produced by
# another team. However, similar functionality could be implemented using the Rasterio
# Python package. If Rasterio is used, it would eliminate the need for GDAL binaries
# and this system path specification.
#--------------------------------------------------------------------------------------

os.environ['PATH'] += ':/Users/billk/miniforge3/envs/py39-pt/bin/'
```

Conda virtual environment