

Step 1: Installing Python

You'll need to make sure you have the necessary python libraries. The algorithm is implemented in Python version 2.7, and will not work with Python 3. The Anaconda distribution of Python is recommended, but any distribution with the appropriate packages will work. You can download Anaconda, version 2.7, here: <https://www.continuum.io/downloads>

Required Packages

Anaconda comes bundled with all of the required packages except GDAL. A full list of required packages is included below.

gdal v2.0+	numpy	scipy	h5py
skimage	sklearn	matplotlib	Tkinter

Step 2: Installing GDAL

Mac and Linux Instructions

- 1) Open the terminal application
- 2) Make sure your path variable is set to point at the anaconda install location. If you installed Anaconda in Step 1, this should already be done. You can test this by typing:

```
conda info
```

If this returns information about the anaconda install, you're good to go. Otherwise (i.e. it returns something similar to "-bash: conda: command not found", find the install location of anaconda and add that directory to the path variable. If you installed anaconda in </Users/your_name/anaconda>, you would enter:

```
export PATH=/Users/your_name/anaconda/bin:$PATH
```

- 3) Type: "conda install gdal" and press enter.
 - a. If this fails due to administrative privileges, type:

```
sudo conda install gdal
```
 - b. This will prompt you for the administrator password; enter it.
- 4) Once the install is complete, enter:

```
export GDAL_DATA=/Users/your_name/anaconda/share/gdal
```

assuming you installed anaconda to </Users/your_name/anaconda>.

Windows Instructions

- 1) Open Anaconda Prompt. This will have been installed as part of Anaconda.
- 2) Type: "conda install gdal" at the command line.

- a. Note that you will likely need administrative privileges for this step. This can be accomplished by right clicking the Anaconda Prompt icon and selecting “Run as Administrator”. Alternatively, find the Anaconda install directory, right click on the folder, select properties and click the security tab. Highlight "Users\..." in the upper pane, and check full control" in the second. Select Apply.
- 3) The program should download the gdal components and install.

Step 3: Creating an Appropriate File Structure

If you are comfortable running scripts from the command line and you know the directory of the images you wish to process, you can skip to Step 4.

The example below will demonstrate creating an appropriate file structure that can help to get the process running quickly. We will use specific directory names for demonstration, though the file and folder names and locations are not required to be the same as below.

1. Set up a folder on your computer to contain the code and all images you are interested in processing. If you are on Windows, we will use the path ‘C:\Image_Processing’. If you are on a Mac, we will demonstrate with ‘/Users/name/Image_Processing’ (note: Replace ‘name’ with the appropriate name for your system).
2. Place the file containing the image processing scripts into this folder.
 - a. If you downloaded the scripts from github, these should be contained in a folder named “OSSP”.
 - b. If you were provided a .zip file containing the code, extract this file here.
3. There should be three training dataset files included with the code:
 - a. icebridge_training_dataset.h5
 - b. wv02_training_dataset.h5
 - c. pan_training_dataset.h5

As their names suggest, each of these files contain training data for a specific type of image.

4. Place the images you wish to process into a subdirectory. On Windows, place the files in ‘C:\Image_Processing\Images’, on Mac: ‘/Users/name/Image_Processing/Images’.

Step 4: Processing the Imagery

The easiest way to get started with the algorithm is to use the ‘oss_p_process.py’ script to analyze all of the imagery in a given folder using the provided training datasets. In the advanced section, we will explore methods for creating custom training datasets and processing an image one step at a time.

1. Windows: Open the Anaconda Prompt that is installed with Anaconda
Mac: Open the Terminal app

2. Change directory to the location of the script. Following from the example above, type:

Windows: `cd C:\Image_Processing\OSSP`

Mac: `cd /Users/name/Image_Processing/OSSP`

3. Run the script to process the images you placed in the 'Images' folder. Following our example from step 3, enter on one line:

Windows:

```
python ossp_process.py C:\Image_Processing\Images srgb
.\training_datasets\icebridge_v3_training_data.h5 -ve
```

Mac:

```
python ossp_process.py /Users/name/Image_Processing/Images srgb
./training_datasets/icebridge_v3_training_data.h5 -ve
```

Understanding the arguments of the `ossp_process.py` script can permit the user to generate commands that process images of different types, in different locations, and using different training datasets. The following shows a generic script call:

```
python ossp_process.py <input_directory> <image_type>
<training_data_file> --splits --parallel
```

Required Arguments:

- **input directory:** directory containing all of the images you wish to process
Note that all .jpg and .tif images in the input directory as well as all sub-directories of it will be processed.
- **image type:** the type of imagery you are processing
 - a. options: {'srgb', 'wv02_ms', 'pan'}
respectively representing RGB imagery taken by a typical camera, Worldview 2 multispectral imagery, and high resolution panchromatic imagery.
- **training dataset file:** complete filepath of the training dataset you wish to use to analyze the input imagery

Optional Arguments:

- **-s | --splits:** The number of times to split the input image for improved processing speed. This is rounded to the nearest square number. Default = 9.
- **-p | --parallel:** The number of parallel processes to run (i.e. number of cpu cores to utilize). Default = 1.
- **-v | --verbose:** Print progress updates as code executes.
- **-e | --extended_output:** Save an rgb image and .csv file of classification results.
- **--training_label:** the label of a custom training dataset. See advanced section for details. Default = **image type**.

In general, images should be split into segments small enough to easily load into RAM. This depends strongly on the computer running these scripts. Segments should typically not exceed 1 or 2gb in size for best results. For 5-7mb RGB images, segmentation is not required (use `-s 1` as the optional argument). For a full multispectral Worldview scene, which may be 16gb or larger,

9 or 16 segments are typically needed. The number of parallel processes to run is up to the user, but should be less than the number of cores on the machine.

Advanced Usage

Creating your own training dataset:

The relevant script for this process is "training_gui.py". Here's how it works:

Place the images you want to use to create your new training set in their own folder. For illustration, we will use the directory: "C:\Image_Processing\Training_Images". We will also assume you are using sRGB images (if not, substitute the appropriate `image_type` argument).

In the same way that you ran the batch process script, type: `python training_gui.py C:\Image_Processing\Training_Images srgb -m 1`

This should bring up a welcome window. Click on the 'srgb' button to begin.

Note: If the number next to `srgb` is not 0, that means the script found an existing training set. You can either continue with the existing one (adding new data) or create a new set by entering a name in the text box and pressing enter. If you create a new dataset with name "elephants" (for example), you will need to add `--training_label elephants` when you call the `ossp_process.py` script

`-m` is short for `--mode`, and is how you select between creating a training dataset and assessing the accuracy of a classified image. For now, you can ignore the second mode of this script. Quotation marks around the directory are needed if there are any spaces in the folder names.

Coming Soon!

- Individual use of the scripts
- More detailed instructions for using `training_gui.py`

Supplement:

Supported imagery formats:

- 1) sRGB
- 2) Panchromatic
- 3) 8-band pansharpened (i.e. WorldView 2)

DISCLAIMER:

The algorithm has been tested with `.jpg` and `.tif(f)` formatted files only. **`ossp_process.py` will only work with folders of `.jpg` and `.tif(f)` files.** Any imagery format readable by GDAL should work when processing images individually, though others formats have not been tested.