A Convolutional Neural Network based high-throughput image classification pipeline - code and documentation to process plankton underwater imagery using local HPC infrastructure and NSF's XSEDE

Moritz S Schmid[1*], Dominic Daprano[2], Kyler M Jacobson[2], Christopher M Sullivan[2], Christian Briseño-Avena[1], Jessica Y Luo[1], Robert K Cowen[1]
[1]Hatfield Marine Science Center, Oregon State University, Newport, OR, USA
[2]Center for Genomic Research and Biocomputing, Oregon State University, Corvallis, OR, USA
*schmidm@oregonstate.edu

Abstract

Scientific imaging (e.g., satellites looking at ocean color, medical imaging) can produce vast quantities of data that need to be processed on time frames similar to data collection. While satellite imaging has many advantages, the satellite's sensors cannot penetrate the ocean's surface more than a few meters. To that effect, underwater imaging systems have been developed in the last 40+ years that can image organisms in-situ in hundreds of meters of water. Underwater imaging systems include those designed for benthic studies (e.g., corals) as well as instruments that document the pelagic realm (e.g., plankton and fish). As an example, we use the In-situ Ichthyoplankton Imaging System (ISIIS) which collects upwards of 14 million images per hour of deployment; in highly productive waters this number can increase up to ten-fold. A typical cruise consisting of 70 hours of ISIIS deployment can yield upwards of 1 billion images of plankton and particles. This big data problem can only be solved by using a high throughput processing pipeline that can be scaled down or up depending on the available resources. Thus, we designed a modular Python-based pipeline that can be deployed on local high-performance computing (HPC) infrastructure such as a University's HPC, as well as on cloud providers. The code provided with this documentation was optimized for Oregon State University's Center for Genomic Research and Biocomputing (CGRB) as well as for the National Science Foundation's Extreme Science and Engineering Discovery Environment (XSEDE), but can easily be adapted to the user's needs. This code and documentation enable 1) the training of a sparse Convolutional Neural Network (sCNN), and 2) applying the sCNN in a processing pipeline to classify all remaining images in an automated fashion. Standard size measurements of the plankton and particles on the segmented images are also taken as part of the pipeline. The pipeline is optimized for speed and can classify upwards of 30 million images per hour on XSEDE Comet GPU compute nodes. End-to-end processing of 1 hour worth of raw imagery data (ca. 14 million images) using XSEDE CPU and GPU nodes takes ca. 2.4 hours, including data upload, segmentation, classification, and obtaining standard length measurements. This enables us to process a typical cruise of ten 7h transects in about a week. A training library of images as well as a video test dataset are supplied with the code. While the pipeline was built for ISIIS images, imagery from other underwater systems and other areas of science can be used with the pipeline.

Cite as
Schmid MS, Daprano D, Jacobson KM, Sullivan CM, Briseño-Avena C, Luo JY, Cowen RK. 2021. A Convolutional Neural Network based high-throughput image classification pipeline - code and documentation to process plankton underwater imagery using local HPC infrastructure and NSF's XSEDE. [Software]. Zenodo. http://dx.doi.org/10.5281/zenodo.4641158

Table of Contents

Complete file list

General files

plankline.py: This is the main pipeline script that deals with managing the starting of all other scripts in the pipeline.
classList.sh: Takes the symbolic link for the training dataset and creates the classList file that is referenced by isiis_scnn during inference and training. This script's location is the SCNN/Data/plankton folder.
xsede_train_scnn.sh: Sets up and streamlines training on XSEDE Comet.
cgrb_train_scnn.sh: Sets up and streamlines training on the CGRB HPC.
pull_images.py: This tool can be used to extract images from certain classes based on the classification output files (.csv). This can be helpful when building a new training dataset or for validation.

Example files

classList: ClassList file created by classList.sh
cgrb.ini: Configuration file for CGRB
xsede.ini: Configuration file for XSEDE Comet
Example training library: Can be found in example_training_library folder
Example video files for testing: Can be found in example_videos folder

Segmentation files

isiis_seg_ff: This is the segmentation x86 binary file.
segmentation.py: Facilitates the segmentation part of the pipeline. This script uses Python's multiprocessing module to create several instances of the seg_ff script and the measurement script. It also moves all of the newly created files.

measure_parallel.py: Provides functionality for measuring the specimen on the image by using skiimage functions. Is used as part of seg_and_compress.py but can also be used as standalone for measuring segments.
xsede_segmentation.sh: This script allocates the CPU resources on XSEDE, sets up the environment for segmentation, and calls the segmentation.py script with the appropriate path.
cgrb_segmentation.sh: This script sets up the environment for segmentation on a local HPC (i.e., the CGRB) and calls the segmentation.py script.
local_segmentation.sh: Sets up the segmentation script to be run locally, i.e, on a personal workstation where no submission is needed.

Classification files

isiis_scnn: This is the compiled binary that does the classification. We have compiled versions for ppc64le and x86 that are available in their corresponding directories in the repository.
classification.py: This is analogous to segmentation.py but for running multiple instances of isiis_scnn on different GPUs. This uses a multiprocessing module and a queue to manage the availability of GPU resources.
split_xsede_classification.sh: This rsyncs the files from the CGRB to XSEDE Comet then splits the files in equal portions and runs xsede_classification.sh on each of the groups of tared images. This is a necessary step for us due to 48 hour time limitations on XSEDE Comet.
xsede_classification.sh: This script is run by sbatch to check out the GPU resources that are used for the rest of the pipeline on XSEDE Comet. Transfers the tared images to the machine scratch space, sets up the environment variables, and then runs classification.py.
cgrb_classification.sh: This script sets up the environment on the CGRB, transfers the tared images to the machine scratch space, and then runs classification.py.
local_classification.sh: Sets up the environment to run isiis_scnn binary on a local workstation.

Introduction

The ISIIS high-resolution imaging system was designed to image large volumes of water (175 L/s) to accurately quantify rare meso-zooplankton such as larval fishes and gelatinous zooplankton in situ (Cowen & Guigand 2008), but it also images smaller plankters including protists and common metazooplankton such as copepods (Fig. 1). High-frequency line-scan cameras enable ISIIS to be towed at 2.5 knots, building a continuous high resolution image as ISIIS is towed. Environmental sensors (e.g., oxygen, temperature) record the conditions the organisms are living in. ISIIS is capable of simultaneous, quantitative sampling of the very fine-scale distributions and sizes of individual plankters ranging from larval fishes, gelatinous and other mesozooplankton, down to their associated prey communities, while doing so over long distances (>100km).

Training the sCNN

A core part of the pipeline is the sparse Convolutional Neural Network (sCNN) as detailed in Luo et al. (2018). Training this sCNN is a crucial step and requires a training library of images to be in place (Fig. 1). The goal is to train the sCNN until the error rate associated with the epochs reaches a plateau. The weights of that epoch can then be used in the pipeline to identify all images in an automated fashion. The training library should be located in a directory where subfolders are named according to the taxa or classes found in the overall imagery. An equally distributed training library usually works best (i.e., approximately the same sample size of training images per class). The reality in oceanography is that sample sizes are often heavily skewed due to some taxa being very abundant but many being quite rare (e.g., larval fish). When rare taxa are of high interest, data augmentation can help to increase the sample size of rare classes (Luo et al. 2018).

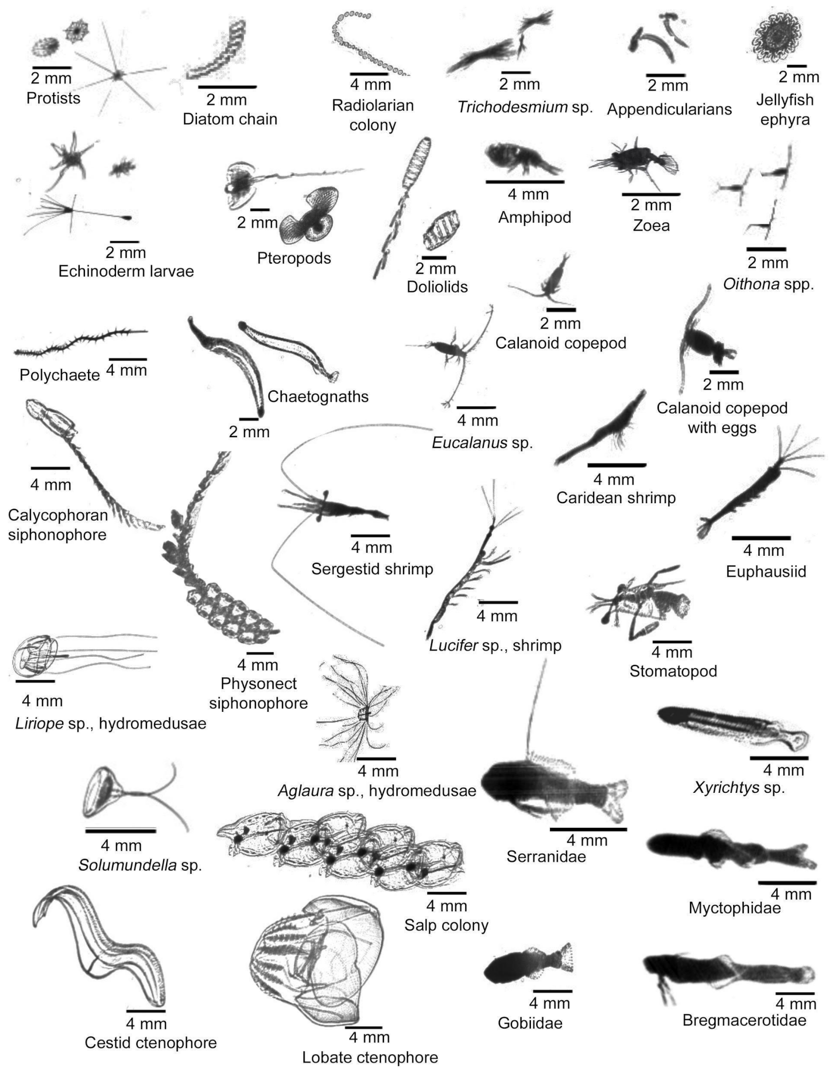


Figure 1. Plankton imaged with the *In-situ* Ichthyoplankton Imaging System (ISIIS) in the Straits of Florida. ISIIS images show representative phyto-, zoo-, and ichthyoplankton taxa making up the training library for the study region (reproduced from Schmid et al. 2020).

isiis_scnn parameters

start: starting epoch number - The starting epoch for training
stop: stopping epoch number - The final epoch to train to
bs: batch size - Indicates how many images are looked at before weights are updated
cD: cuda device - Specifies the cuda device to run the isiis_scnn on
unl: image directory - Directory containing the images to be segmented
ilr: Initial learning rate – Changes how quickly or slowly the neural network adjusts to training data
lrd: Learning rate decay - Adjusts how the learning rate changes between epochs.
vsp: Validation percentage - Percentage of the training library data that you want to use as validation dataset.

Argument notes

If the start epoch equals stop epoch no training will occur; this is used exclusively for classification in the pipeline. The setting can be changed in classification.py.

vsp reserves a portion of the training set for testing purposes. Initially a 20% validation set can be used for training. After fine-tuning, the CNN can be re-trained without the validation data (i.e., validation_percent = 0) to leverage the maximum available data.

During each epoch a new learning rate is calculated, using the formula: learning_rate = initial_learning_rate * exp(-learning_rate_decay * epoch).

Compilation

Dynamically compiled versions of isiis_scnn for ppc64le and x86 architectures are supplied. To run the isiis_scnn binaries all the necessary libraries have to be added to the LD_LIBRARY_PATH environment variable. This project makes use of autotools to simplify the process of setting up the necessary dependencies, refer to the pipeline setup guide for details.

General training setup

An interactive session on a GPU machine has to be initiated. At the local HPC infrastructure of Oregon State University, the Center for Genomic Research and Biocomputing (CGRB), this can be done using SGE. To start, a qrsh link to a GPU machine such as ibm-power3 has to be established. Please reference this guide for more information on SGE.

```
$ qrsh -q ibm-cgrb@ibm-power3
```

The directory for the isiis_scnn instance has to be created. This directory is going to be referred to as SCNN and needs to be moved to the fast SSD storage (i.e., scratch space) on the machine in use, on the CGRB HPC this is mounted at /data.

```
$ mv SCNN/data/
```

There are two main directories, the weights directory and the data/plankton directory. The weights directory contains all of the weight files resulting from each epoch of training. Old weights should be removed before starting to train.

```
$ rm -f SCNN/weights/*
```

The data/plankton directory contains information about the training dataset. This is done through the train symbolic link (it is advised to remove the old symbolic links beforehand). Use the supplied example training library to test this step and training. While a test symbolic link has to be set up too, the sCNN pulls images for out of bag testing from the training folder, thus the test folder can remain empty.

```
$ ln –s /data/<training_dataset_dir>/  train
```

The train link has to point to a directory that has subdirectories named for the classes of images in the training library. For example, the <train_dataset_dir>/crustacean_zoea_crab/ should contain all of the crustacean zoea crab images. It is important that the training dataset is also in the SSD storage of the machine used for training.

Next, the classlist file has to be created. This is done by running the classList.sh script in the data/plankton directory.

```
$ bash classList.sh
```

It should be verified that the classlist contains the correct number of classes (and not a number from previous training). Both of the following commands should return the same number. If they do not, this indicates that directories under “train” have spaces in them; no spaces are allowed.

```
$ wc -l classList
$ ls train/ | wc -l
```

This process is done in an automated fashion through the cgrb_train_scnn.sh and xsede_train_scnn.sh scripts that are detailed below.

Training locally

Before starting the training process, it should be verified that no other processes use that GPU. The nvidia-smi command can be used to check running processes.

The GPU ID of the idle GPU that is to be used should be noted down, as it has to be given as the cuda device parameter in the training command.

To start training, the start epoch has to be set to 0 and the stop epoch has to be set to any positive integer. Choose a lower number for the stop epoch, such as 10 or 20, when running a first test. The amount of epochs needed to get to the error percentage plateau depends among other things on the number of classes in the training library. When training with over 150 training classes a stop epoch of 350-400 can be needed; the error rate should be monitored in order to detect the error rate plateau. If start epoch = stop epoch then no training is initiated and only classification will occur.

```
$ ./isiis_scnn -start 0 -stop <stop_epoch> -bs <batch_size> -cD <cuda_device>   >
scnn_train.log
```

As training is underway, progress and error percentage can be viewed in the log file.

```
$ cat -tail scnn_train.log
```

A script has been created to do this in a more automated fashion at a local HPC. The script is called cgrb_train_scnn.sh and can be found in the scripts directory. At the CGRB HPC it can be run on the ibm-power3 machine using SGE by running the following command:

```
$ SGE_Batch -q ibm-cgrb@ibm-power3 -c ‘bash cgrb_train_scnn.sh -c <SCNN_dir> -t
<train_dir> -e <stop_epoch> -b <batch_size> -d <gpu_device>’ -r <log_dir>
```

NOTE: Since this machine uses the ppc64le architecture, the SCNN directory also needs to contain the isiis_scnn binary for ppc64le.

The script can be modified for a different infrastructure by changing the scratch variable and setting up the proper LD_LIBRARY_PATH for the isiis_scnn binary, as detailed in the isiis_scnn compilation section.

Training on the XSEDE HPC infrastructure

Examples given here are for training isiis_scnn on XSEDE (xsede.org) Comet GPU nodes. Note that XSEDE computational resources are obtained through grant proposals.

Job submission on XSEDE is facilitated by the script xsede_train_scnn.sh. The script allocates a full 4 GPU node to the training task. Even though training the scnn only uses a single GPU, it is necessary to allocate the whole node, since the user needs exclusive access to the SSD storage of the machine.

The sbatch command executes a bash script that contains special configurations marked by lines starting with #SBATCH. These lines are configurations for how the job should be started on XSEDE, i.e., the resources to be requested, output file, and the wall time for the process. More information and examples of these configurations can be found here. This is essentially the same as the SGE mechanism that is used in other places such as the CGRB, it allows users to queue jobs so that they can be run once the resources become available. Here is an example submission:

```
$ sbatch xsede_train_scnn.sh -c <scnn_dir> -t <train_dir> -e <stop_epoch> -b <batch_size> -d
<gpu_device>
```

NOTE: Similar to the local HPC, this script requires that the SCNN directory contains a isiis_scnn binary that works on the infrastructure. In the case of XSEDE this is the x86 binary and for CGRB this is the ppc64le binary.

In order to check the error rate when training on XSEDE, the log file that was specified in the sbatch options at the top of xsede_train_scnn.sh can be accessed.

The script can be modified for a different infrastructure by changing the scratch variable and setting up the proper LD_LIBRARY_PATH for the scnn binary; see details in the compilation section.

Interpreting the training log files

The most important measurements to interpret the log files are the ‘Mistake’ variable, indicating the percentage of wrong classifications per epoch, as well as the negative log likelihood variable associated with the epoch (Table 1).

Table 1. Variable names and explanations used in the sCNN log files.

Variable	**Explanation**
Mistakes	Percentage of wrongly classified images
NLL	Negative log likelihood (smaller is better)
MegaMultiplyAdds/sample	Number of unit operations per pixel, in millions
Time	Seconds elapsed for the current step
GigaMultiplyAdds/s	Number of multiply add operations per second, in billions

Rate	Number of images processed per second

Processing pipeline

The image processing pipeline uses the weights of an epoch that were generated during sCNN training to classify previously unclassified images. It does this by assigning each image *n* probabilities, where *n* is the number of classes in the training library. Each probability reflects the likelihood of an image pertaining to a certain class (probabilities for an image sum up to 1). In order to get there, the pipeline first destacks the AVIs generated during the deployments of ISIIS at sea. Single TIFFs are then flatfielded (i.e., removal of line scan camera artifacts, background), and a k-harmonic means clustering algorithm detects single regions of interest (ROI; i.e., a single plankton or particle specimen), which are then saved as jpegs. Jpegs are then classified by the sCNN (Luo et al. 2018). The pipeline also calculates area and perimeter of the objects in the ROIs, as well as major-, and minor axes lengths, based on the Python scikit-image package. Measurements are in pixels and need to be multiplied by the pixel size of the user's instrument. These measurements can be used for further analyses such as calculating the equivalent spherical diameter which is often used for carbon conversions.

Overview

The pipeline automates the process of running segmentation and classification processes for a batch of videos (often a harddrive pertaining to a transect on which an instrument was deployed). Inside the scripts, groups of video files will be referred to as a drive. To start processing the pipeline expects a directory containing the video files that need to be classified (i.e., /raw). The pipeline will create the remaining directory structure. Use the supplied example video files to test the pipeline.

raw/
segmentation/
measurements/
classification/

These directories are taken as constants; thus, it is important that their names are not changed. When the plankline.py script is run it asks the user to input configurations for the pipeline, through the use of config files (.ini) which are stored for future use. Based on the configurations, segmentation and classification can either take place on a local HPC infrastructure (i.e., CGRB in the examples given) or the remote NSF XSEDE HPC infrastructure.

Due to asynchronous job submissions, the plankline.py script has to be run multiple times; however, the script stores its progress and will pick up where it left off. Because of this, the plankline.py script will create IN_PROGRESS files so that classification is not started until after segmentation finishes.

Configuration files

Configuration files determine how the pipeline is run. The files can be created during runtime of the plankline.py or they can be submitted on the command line with the -c option.

```
$ python3 plankline.py -c xsede.ini
```

The configuration files contain information such as the architecture that the pipeline is run on, the number of processes that are concurrently running segmentation, and the number of classification instances per GPU. The files also contain specifics for the XSEDE HPC, like remote host, remote user, and remote storage path. Example config files can be found under the names xsede.ini and cgrb.ini.

Local HPC vs XSEDE HPC

While running the pipeline locally or running the pipeline remotely does not change the classification or segmentation mechanism, some steps differ, such as setting up the data in the correct places.

For the XSEDE HPC ssh-keys need to be set up so that passwordless login can be done in the script. This is necessary so that data being segmented or classified can be transferred to XSEDE Comet using scripts without prompting for passwords. It should only take small modifications to make the pipeline work on any HPC infrastructure by replacing the job submission mechanism in the XSEDE script with the job submission mechanism specific to the infrastructure the pipeline needs to run on), and pointing to the proper libraries.

Pipeline Setup

In order to set up the pipeline, some configurations have to be made. First, an instance of python3 with all of the necessary modules has to be installed. Next, the shared library files need to be installed. Lastly, FFprobe and FFmpeg need to be installed. FFmpeg and FFprobe split the raw videos into frames used for segmentation. Static builds for these programs can be found here. To check if they have already been installed, the following commands can be run. Reference the FFmpeg and FFprobe documentation for additional usage information.

In order to streamline this process, this pipeline makes use of autoconf.

```
$ autoconf
```

Autoconf will generate a configuration file that is used to check the setup of the current machine and help to assure that the pipeline will not fail later down the line.

```
$ ./configure
```

If ./configure ran properly then it created a makefile.

Running the pipeline

The plankline.py script and any config files have to be copied to the directory containing the raw .avi files. The pipeline can be run via plankline.py, with or without a config file.

```
$ python3 plankline.py -c xsede.ini
```

```
$ python3 plankline.py
```

If no configuration file is specified the pipeline will prompt for any necessary configuration variables. The pipeline then starts running segmentation, and once completed moves to classification.

Note: If prompted, all necessary python modules have to be downloaded. Even though they are not all required in the plankline.py script directly, they will be required in other scripts that are called by plankline.py. If python is in a different location, the python location has to be updated in the scripts.

Pipeline details

Segmentation

A description of scripts used during segmentation can be found here.

The main difference between running segmentation on XSEDE Comet vs the CGRB HPC is the use of the cgrb_segmentation.sh script vs the use of the xsede_segmentation.sh script (Fig. 2). Whichever HPC infrastructure is used, only the setup script needs to be changed, and segmentation.py can be used irrespective of the segmentation location. The segmentation.py script takes the raw AVI files and turns them into a series of jpeg images containing objects needing to be identified. This is done using two main tools. ffmpeg transforms the video into individual frames, and then applies flat fielding to remove line scan artifacts. isiis_seg_ff is a binary that takes the individual frames that have been output from ffmpeg and segments the images into a series of jpegs containing objects to be classified. isiis_seg_ff is only available as a binary file and can only run on x86 architecture.

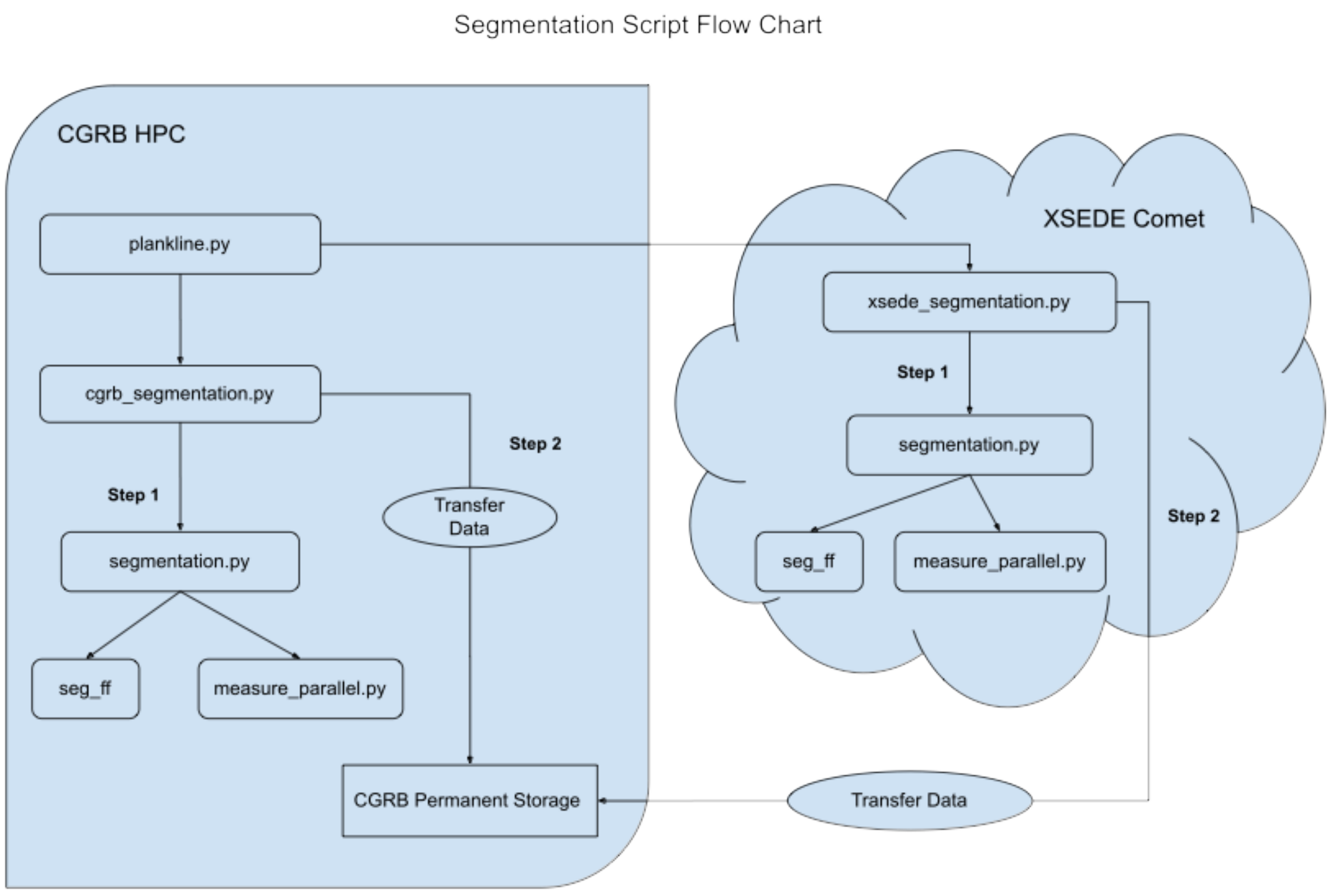


Figure 2. Scripts called during segmentation on a local HPC such as the CGRB and a remote HPC such as XSEDE Comet.

Classification

A description of the files used during classification can be found here.

Similar to segmentation, the classification.py script is used independent of the infrastructure used (Fig. 3). On XSEDE Comet it is necessary to split the classification jobs up into multiple processes in order to conform to the 48 hour processing time limit. The script split_xsede_classification.sh takes care of this. classification.py uses a queue and multiprocessing pool to manage the availability of GPUs and makes sure there is always an instance of the isiis_scnn running on them. plankline.py sets up the environment necessary to run the scnn. The binary that has been pre-built for ppc64le and x86 and performs classification on the images resulting from the segmentation part of the pipeline.

To perform classification, the isiis_scnn_binary is run with the start epoch equal to the stop epoch, where the epoch number corresponds to the number of the weights file found in the SCNN/weights directory.

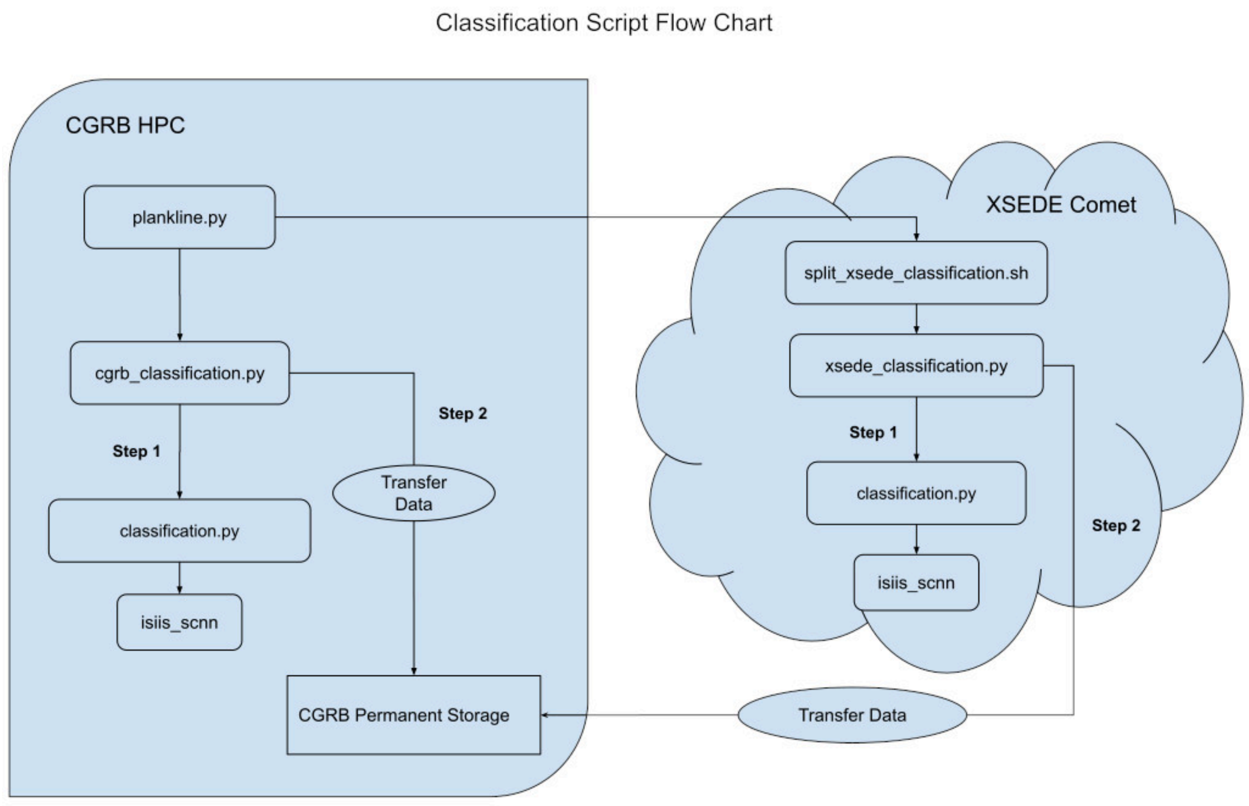


Figure 3. Scripts called during classification on a local HPC such as the CGRB and a remote HPC such as XSEDE Comet.

Machine scratch space

All of the files that are written during processing should be going into the fast machine scratch space (also referred to as SSD storage before). Using scratch space will greatly reduce processing times. The machine scratch space is usually only accessible on a jobs host machine. This means that it will usually be inaccessible through the submit host and needs to be specified in the job's bash script. On XSEDE Comet the machine scratch space for a job with jobid $SLURM_JOBID for the user $USER can be found at /scratch/$USER/$SLURM_JOBID/ once on the machine.

```
$ cd /scratch/$USER/$SLURM_JOBID/
```

At the CGRB the machine scratch space can be found at /data once on the machine

```
$ cd /data
```

Fast file transferring machines

There might be special hosts for transferring files on the user's infrastructure, especially for large quantities. Using the submit host for transferring files can slow down job submissions for other users. On XSEDE Comet the file transfer host is:

```
$ ssh <xsede_username>@oasis-dm-interactive.sdsc.edu
```

At the CGRB the file transfer host is:

```
$ ssh -p <cgrb_port> <cgrb_username>@files.cgrb.oregonstate.edu
```

XSEDE Comet notes

Adjusting slurm batch scripts for different GPUs

Slurm batch scripts are a common way of interacting with HPC infrastructure and are also used at XSEDE Comet. They are essentially bash scripts with special headers that contain information about the resources being requested and details on how jobs must be run. More information about the system can be found here: https://slurm.schedmd.com/sbatch.html

Here are some notes to remember when writing SBATCH commands for different GPU resources. Nvidia K80s requires 6 tasks per GPU when allocating whereas Nvidia P100s require 7 tasks per GPU. The correct values need to be set in the --ntasks-per-node sbatch option.

Example 1: 4 k80 GPUs

The number of tasks per node input is based on the number of tasks per GPU multiplied by the number of GPUs. In the case of the k80 GPU there are 6 tasks per GPU, and 4 GPUs, thus --ntasks-per-node=(4 GPUs * 6 tasks = 24). The following code block needs to be at the top of the bash script.

```
#SBATCH -A osu119
#SBATCH --partition=gpu
#SBATCH -gres=gpu:k80:4
#SBATCH --ntasks-per-node=24
#SBATCH --nodes=1
#SBATCH -t 23:30:00
```

Example 2: 4 p100 GPUs

In the case of the p100 GPU there are 7 tasks per GPU, and 4 GPUs, thus --ntasks-per-node=(4 GPUs * 7 tasks = 28). The following code block needs to be at the top of the bash script.

```
#SBATCH -A osu119
#SBATCH --partition=gpu
#SBATCH -gres=gpu:p100:4
#SBATCH --ntasks-per-node=28
#SBATCH --nodes=1
#SBATCH -t 23:30:00
```

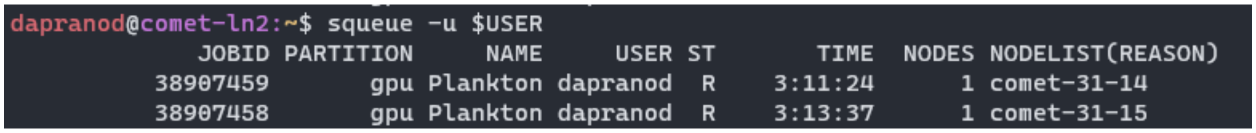
Example 3: Interactive session on K80s

Interactive sessions set up all the same resources as the sbatch, however, instead of executing a script and closing, such a session opens a shell on the machine that the user is then able to use until the allotted time runs out. In this case, all 4 GPUs were allocated on a node for 2 hours and a bash shell was opened for the user to run commands on.

```
$ srun --gres=gpu:K80:4 --ntasks-per-node=24  -t 2:00:00 -A osu119 --pty --wait 0 /bin/bash
```

Checking job progress on XSEDE Comet (SLURM)

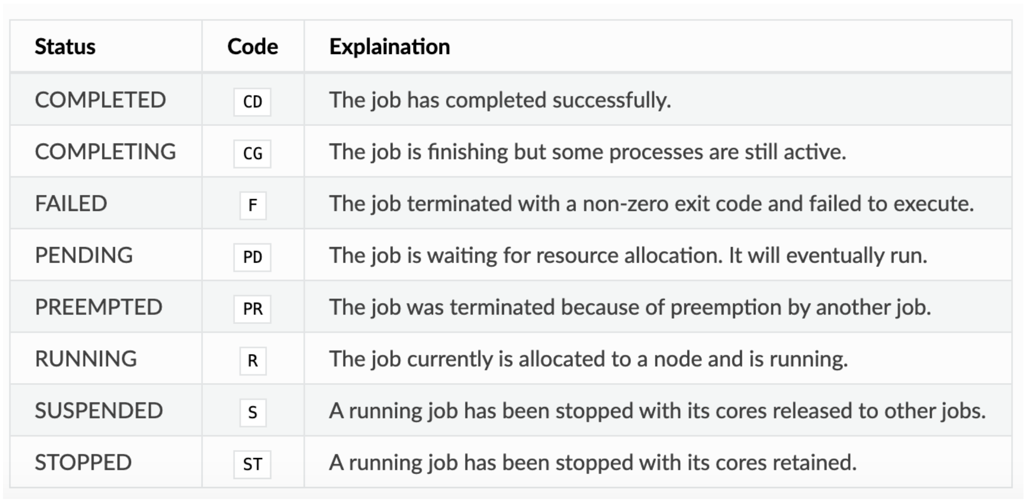
An important part of running the pipeline is checking the progress of submitted jobs and looking out for any potential errors. Job progress on XSEDE Comet can be checked using the squeue (slurm queue) command and the USER environment variable (Fig. 4). This will provide information about the jobs that are under a user and their status.



```
dapranod@comet-ln2:~$ squeue -u $USER
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          38907459       gpu Plankton dapranod  R    3:11:24      1 comet-31-14
          38907458       gpu Plankton dapranod  R    3:13:37      1 comet-31-15
```

Figure 4. Example output of a job progress check on XSEDE Comet.

JOBID**:** This number can be used with scancel (Example: "scancel 36712856", NOTE: Only the user that submitted the job can cancel it)
PARTITION**:** The type of resources that was requested (GPU, CPU)
NAME**:** The name of the job that was started
USER: The account username of the user that submitted the job
NODELIST: This is the host machine that the job is running on. Connect to this machine via SSH (Example: "ssh comet-31-15).
ST: The status of the process. Here is a list of all the possible statuses (Fig. 5).



Status	Code	Explaination
COMPLETED	CD	The job has completed successfully.
COMPLETING	CG	The job is finishing but some processes are still active.
FAILED	F	The job terminated with a non-zero exit code and failed to execute.
PENDING	PD	The job is waiting for resource allocation. It will eventually run.
PREEMPTED	PR	The job was terminated because of preemption by another job.
RUNNING	R	The job currently is allocated to a node and is running.
SUSPENDED	S	A running job has been stopped with its cores released to other jobs.
STOPPED	ST	A running job has been stopped with its cores retained.

Figure 5. Status codes on XSEDE Comet.

Common issues and errors

isiis_seg_ff segmentation fault

If the output directory (-o) for the segmented images is too long then seg_ff will throw a segmentation fault. For example, this output path: "/scratch/dapranod/36712856/129/segmentation" would not work, while this one: "/scratch/dapranod/36712856/129/s" would.

Image extraction tool

This tool lets the user extract classified images based on criteria such as minimum classification percentage assigned and classified taxon. The main application of the tool is to extract images and then compare the computer-generated classifications with the expert visual identifications of the images for confusion matrices.

What it does

The tool uses command line options in order to find the images that were classified as being a certain taxon by looking through the output csv files from classification and then pulling these images from their corresponding tar file.

Command line options

These can be accessed on the command line by typing “python3 pull_images.py -h”

Usage

```
$ python3 pull_images.py [-h] -t TAXON [TAXON ...] (-p PROBABILITY_TAXON | -b) (-d
INPUT_DRIVE | -c INPUT_CSV) -o OUTPUT -r RAW_TARS [-m MIN_IMAGES]
```

Arguments

-h, --help

Show the usage message then exit

-t TAXON [TAXON ...], --taxon TAXON [TAXON …]

The taxa that the script should search for in the csv files. This should be given as a list of taxa separated by a space. Full taxon names can be written out, or multiple taxa can be searched by using a more generic string. E.g, writing “fish” will find all taxa with “fish” in the name. It is more efficient to search for multiple taxa at a time since untarring the image folders is what takes most of the time.

-p PROBABILITY_TAXON, --probability_taxon PROBABILITY_TAXON

Extracts all images for a taxon over the threshold probability (value between 0 and 1). Note: If the search string for taxa is ‘fish’ and a threshold of 0.2 is set, an image that was classified as fish_a with 0.22 probability and fish_b with 0.21 probability would be extracted in searches for both taxa. This option is useful when building new training sets as it can return images that were subjected to false negative classification (i.e., second highest probability).

-b, --best_taxon

Extracts the image only if the search for taxon pertains to the highest probability given to the image (i.e., an image will only be extracted once and only if the taxon search string coincides with the highest probability given).

Even though the --probability option will usually extract more images than the --best_taxon option, it takes less time to run. This is because the --probability option does fewer comparisons on the probabilities. The --probability option will slow down as more taxa are added.

See timing example for more info.

-d INPUT_DRIVE, --input_drive INPUT_DRIVE

The directory containing the csv files.

-c INPUT_CSV, --input_csv INPUT_CSV

The csv file that should be searched for taxa and images. Either the --input_csv option or the --input_drive option can be used. Both options will yield the same output file structure, --input_drive will work on all csv files in a folder.

-o OUTPUT, --output OUTPUT

The output directory where folders and images should be copied to. This folder must be empty.

-r RAW_TARS, --raw_tars RAW_TARS

The directory of all of the tared images that correspond to the input csv files.

-m MIN_IMAGES, --min_images MIN_IMAGES

The minimum number of images needed to untar a folder, this defaults to 0. The user might want to decide to ignore tared folders that only contain one image of interest due to the time it takes to untar.

-dc, --different_columns

If the colums (taxon names) are different in some csv files, this option will prevent the program from terminating.

-s, --strict_subclasses

Using this flag will make the script match the input class names exactly and will no longer match any class that contains the substring.

Data input

The input csv files must all have the same structure, i.e. they must all have the same taxon in each column of the csv files, unless the --different_columns option is used.

The tar file directory that is used as input must have the same datetime stamp as the csvs, this is usually guaranteed if the tar and csv files from pipeline output are used. The program uses regex to get the numeric date string from the csv file names and then uses this to find the correct tar file with that same numeric datetime in the tar directory.

Example:
"20180706170805.476-library_bigcam.csv" - file for the classification information
"20180706170805.476.tar.gz" - file containing the images.

Example usage

To find all images belonging to protists and fish from the csvs in the folder "excsvs", with the corresponding tars at the "extars" location, use:

```
$ python3 pull_images.py -p .2 -d excsvs/ -o exoutMulti3/ -r extars/ -t protist fish
Finding the sub_taxon:
0. protist_acantharia
1. protist_dark_center
2. protist_fuzzy_olive
3. protist_noctiluca
4. protist_noctiluca_long_flagella
5. protist_other
6. protist_radiolaria
…
10. fish_ ...
```

Timing examples

Single taxon example

When using the -p, --probability option with a threshold of .2, it took 1 minute 58 seconds vs the 4 minutes 17 seconds it took with the -b, --best_taxon option.

-p, --probability option

```
$ time python3 pull_images.py -p .2 -d excsvs/ -o exoutMulti5/ -r extars/ -t protist_fuzzy_olive
Input Drive: excsvs/
Number of Input CSVs: 4
Raw tar Directory: extars/
Output Directory: exoutMulti5/
Minimum images needed to unzip: 0
----- Finding chosen taxon with probabilities above 0.2 -----

Finding the sub_taxon:
0. protist_fuzzy_olive
Finding images from excsvs/20180706190732.329-library_bigcam.csv
Found 1 images in excsvs/20180706190732.329-library_bigcam.csv
Unzipping extars/20180706190732.329.tar.gz into exoutMulti5/
Building file structure
Finding images from excsvs/20180706170805.476-library_bigcam.csv
Found 220 images in excsvs/20180706170805.476-library_bigcam.csv
Unzipping extars/20180706170805.476.tar.gz into exoutMulti5/
Building file structure
Finding images from excsvs/20180706200311.924-library_bigcam.csv
Found 146 images in excsvs/20180706200311.924-library_bigcam.csv
Unzipping extars/20180706200311.924.tar.gz into exoutMulti5/
Building file structure
Finding images from excsvs/20180706191023.549-library_bigcam.csv
Found 0 images in excsvs/20180706191023.549-library_bigcam.csv
----- Done -----
113.078u 0.560s 1:58.51 95.8%   0+0k 0+46976io 0pf+0w
```

-b, --best_taxon option

```
$ time python3 pull_images.py -b -d excsvs/ -o exoutMulti5/ -r extars/ -t protist_fuzzy_olive
Input Drive: excsvs/
Number of Input CSVs: 4
Raw tar Directory: extars/
Output Directory: exoutMulti5/
Minimum images needed to unzip: 0
----- Finding chosen taxon where they are the top probability -----

Finding the sub_taxon:
0. protist_fuzzy_olive
Finding images from excsvs/20180706190732.329-library_bigcam.csv
Found 1 images in excsvs/20180706190732.329-library_bigcam.csv
```

```
Unzipping extars/20180706190732.329.tar.gz into exoutMulti5/
Building file structure
Finding images from excsvs/20180706170805.476-library_bigcam.csv
Found 140 images in excsvs/20180706170805.476-library_bigcam.csv
Unzipping extars/20180706170805.476.tar.gz into exoutMulti5/
Building file structure
Finding images from excsvs/20180706200311.924-library_bigcam.csv
Found 83 images in excsvs/20180706200311.924-library_bigcam.csv
Unzipping extars/20180706200311.924.tar.gz into exoutMulti5/
Building file structure
Finding images from excsvs/20180706191023.549-library_bigcam.csv
Found 0 images in excsvs/20180706191023.549-library_bigcam.csv
----- Done -----
252.585u 0.799s 4:17.27 98.4%   0+0k 0+28672io 0pf+0w
```

Multiple taxa example

When using the -p, --probability option with a threshold of .2, it took 4 minute 56 seconds vs the 6 minutes 55 seconds it took with the -b, --best_taxon option.

-p, --probability option

```
$ time python3 pull_images.py -p .2 -d excsvs/ -o exoutMulti3/ -r extars/ -t protist fish
Input Drive: excsvs/
Number of Input CSVs: 4
Raw tar Directory: extars/
Output Directory: exoutMulti3/
Minimum images needed to unzip: 0
----- Finding chosen taxon with probabilities above 0.2 -----

Finding the sub_taxon:
0. protist_acantharia
1. protist_dark_center
2. protist_fuzzy_olive
3. protist_noctiluca
4. protist_noctiluca_long_flagella
5. protist_other
6. protist_radiolarian
7. fish_bregmacerotidae
8. fish_carangidae
9. fish_ceratioidei_or_tetraodontidae
```

```
10. fish_echeneidae
11. fish_engraulidae
12. fish_gobiidae
13. fish_gonostomatidae
14. fish_labroidei
15. fish_leptocephali
16. fish_microdesmidae
17. fish_myctophidae
18. fish_ophidiidae
19. fish_phosichthyidae
20. fish_pleuronectiformes
21. fish_scombridae
22. fish_serranidae
23. fish_stocky
24. fish_trichiuridae
25. fish_unknown
26. fish_xyrichtys
Finding images from excsvs/20180706190732.329-library_bigcam.csv
Found 6032 images in excsvs/20180706190732.329-library_bigcam.csv
Unzipping extars/20180706190732.329.tar.gz into exoutMulti3/
Building file structure
Finding images from excsvs/20180706170805.476-library_bigcam.csv
Found 5434 images in excsvs/20180706170805.476-library_bigcam.csv
Unzipping extars/20180706170805.476.tar.gz into exoutMulti3/
Building file structure
Finding images from excsvs/20180706200311.924-library_bigcam.csv
Found 3927 images in excsvs/20180706200311.924-library_bigcam.csv
Unzipping extars/20180706200311.924.tar.gz into exoutMulti3/
Building file structure
Finding images from excsvs/20180706191023.549-library_bigcam.csv
----- Done -----
244.793u 4.439s 4:56.20 84.1%   0+0k 1661568+2903040io 0pf+0w
```

-b, --best_taxon option

```
$ time python3 pull_images.py -b -d excsvs/ -o exoutMulti4/ -r extars/ -t protist fish
input Drive: excsvs/
Number of Input CSVs: 4
Raw tar Directory: extars/
Output Directory: exoutMulti4/
Minimum images needed to unzip: 0
----- Finding chosen taxon where they are the top probability -----
```

```
Finding the sub_taxon:
0. protist_acantharia
1. protist_dark_center
2. protist_fuzzy_olive
3. protist_noctiluca
4. protist_noctiluca_long_flagella
5. protist_other
6. protist_radiolarian
7. fish_bregmacerotidae
8. fish_carangidae
9. fish_ceratioidei_or_tetraodontidae
10. fish_echeneidae
11. fish_engraulidae
12. fish_gobiidae
13. fish_gonostomatidae
14. fish_labroidei
15. fish_leptocephali
16. fish_microdesmidae
17. fish_myctophidae
18. fish_ophidiidae
19. fish_phosichthyidae
20. fish_pleuronectiformes
21. fish_scombridae
22. fish_serranidae
23. fish_stocky
24. fish_trichiuridae
25. fish_unknown
26. fish_xyrichtys
Finding images from excsvs/20180706190732.329-library_bigcam.csv
Found 3809 images in excsvs/20180706190732.329-library_bigcam.csv
Unzipping extars/20180706190732.329.tar.gz into exoutMulti4/
Building file structure
Finding images from excsvs/20180706170805.476-library_bigcam.csv
Found 3819 images in excsvs/20180706170805.476-library_bigcam.csv
Unzipping extars/20180706170805.476.tar.gz into exoutMulti4/
Building file structure
Finding images from excsvs/20180706200311.924-library_bigcam.csv
Found 2799 images in excsvs/20180706200311.924-library_bigcam.csv
Unzipping extars/20180706200311.924.tar.gz into exoutMulti4/
Building file structure
Finding images from excsvs/20180706191023.549-library_bigcam.csv
Found 4354 images in excsvs/20180706191023.549-library_bigcam.csv
```

```
Unzipping extars/20180706191023.549.tar.gz into exoutMulti4/
Building file structure
----- Done -----
391.612u 3.329s 6:55.67 95.0%   0+0k 0+1891968io 0pf+0w
```

References

Cowen RK, Guigand C. 2008. In Situ Ichthyoplankton Imaging System (ISIIS): System design and preliminary results. Limnol Oceanogr Meth 6:126-32 https://doi.org/10.4319/LOM.2008.6.126

Luo JY, Irisson J-O, Graham B, Guigand C, Sarafraz A, Mader C, Cowen RK. 2018. Automated plankton image analysis using convolutional neural networks. Limnol Oceanogr Methods 16: 814– 827 https://doi.org/10.1002/lom3.10285

Schmid, MS, Cowen, RK, Robinson, K, Luo, JY, Briseño-Avena, C, Sponaugle, S. 2020. Prey and predator overlap at the edge of a mesoscale eddy: fine-scale, in-situ distributions to inform our understanding of oceanographic processes. Sci Rep 10:921 https://doi.org/10.1038/s41598-020-57879-x

Acknowledgements

We thank Manu Shantharam from XSEDE ECS for his support. This project was funded by the National Science Foundation under grant numbers OCE-1737399 and OCE-1419987, the National Aeronautics and Space Administration under grant number 80NSSC20M0008, the Belmont Forum (through NSF grant number 1927710), as well as the Extreme Science and Engineering Discovery Environment (XSEDE) under grant number OCE170012.

Disclaimer and License