UAF-Plankline

v1.0

Generated by Doxygen 1.9.1

Chapter 1

1. Recommended Installs

These are common utilities and/or prerequisites for one or more of the steps recommended by this setup document.

```
sudo apt-get install r-base net-tools ethtool libopencv-dev libgdal-dev timeshift htop openssh-server unzip gi
```

Python Related: We are using python3 throughout the plankline processing scripts, and need to install some additional pacakges/libraries for image processing.

```
sudo apt-get install python3 python3-devpython3-pip python3-skimage python3-opencv
pip3 install gpustat
```

NVIDIA Requirements for GPGPUs: With these commands we first install the app key used to authenticate the nvidia package that we download using wget. The following commands then install the pacakge *cuda keyring*, which allows us to download and install nvidia drivers directly. Finally we install the development tools needed to compile CUDA applications ourselves.

```
sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/3bf86
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-keyring_1.0-1_all.deb

sudo dpkg -i cuda-keyring_1.0-1_all.deb
sudo apt-get update
sudo apt-get install libsparsehash-dev cuda-nvcc-11-8 libcublas-11-8 libcublas-dev-11-8
```

1.0.1 1.1 Compile and Install OpenCV 3.4

Install the prerequisites

```
sudo apt-get install build-essential cmake libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-
sudo apt-get install libjpeg-dev libpng-dev libtiff5-dev jasper libdc1394-dev libeigen3-dev libtheora-dev libv
```

Get opencv2 and install it

```
cd /opt

git clone --branch 3.4 --single-branch https://github.com/opencv/opencv.git
git clone --branch 3.4 --single-branch https://github.com/opencv/opencv_contrib.git

cd opencv
mkdir release
cd release

cmake -D BUILD_TIFF=ON -D WITH_CUDA=ON -D WITH_OPENGL=OFF -D WITH_OPENCL=OFF -D WITH_IPP=OFF -D WITH_TBB=ON -D

make -j
sudo make install

sudo ldconfig
pkg-config --modversion opencv
```

1.1 2. NVIDIA CUDA

```
sudo apt-get install nvidia-cuda-toolkit nvtop
```

Nvidia Drivers for Tesla cards:

```
sudo apt-get install libnvidia-compute-470 nvidia-utils-470 nvidia-driver-470-server
sudo ldconfig
```

Nvidia Drivers for GTX cards:

TODO: Add link to table of driver versions for each card.

```
sudo apt-get install libnvidia-compute-515 nvidia-utils-515 nvidia-driver-515
sudo ldconfig
```

You can monitoring active NVIDIA processes with these commands

```
nvidia-smi
nvtop
```

1.1.1 Install RStudio Server

Rstudio Server is a useful tool for running R processing scripts directly on the plankline computers using your local computer's browser.

```
sudo apt-get install gdebi-core;
wget https://download2.rstudio.org/server/jammy/amd64/rstudio-server-2022.07.1-554-amd64.deb;
sudo gdebi rstudio-server-2022.07.1-554-amd64.deb;
sudo nano /etc/rstudio/rserver.conf;
```

Add "www-port=80", then restart the service:

```
sudo systemctrl restart rstudio-server
```

The Rstudio Server should now be available at http://<computer name>/. For example, http://plankline-2/. Please see the section below about setting the computer name is it has not already been set.

To install the plankline processing scripts: Note, that the default "home" directory for rstudio will be the home directory of the user that you are log in as.

```
cd ~
git clone https://github.com/tbrycekelly/Plankline-Processing-Scripts.git
```

From an Rstudio Server instance (i.e. in the browser), open the Plnkline-Processing-Scripts project, and nvigate to the Workflows subdirectory. This is the folder that contains all the basic workflow scripts for processing and analyzing the plankline image and sensor data. Run the *one time setup* script to install and verify functionality. This only needs to be done once per install, but it is an easy cehck that everything is working as it should.

1.1.2 Install Netdata

Netdata is a useful system monitor that is accessible over the network (e.g. http://<computer name>↩ :19999/). It gives detailed statistics about a wide variety of computer resources (except for the GPUs).

```
wget -O /tmp/netdata-kickstart.sh https://my-netdata.io/kickstart.sh && sh /tmp/netdata-kickstart.sh
```

Point a broswer at http://<computer name>:19999/ to verify install.

1.2 Linux Reference Guide

1.2.1 1. Introduction to working with Linux

1.2.2 2. Common commands and activities

Set password:

```
sudo passwd plankline
```

Set hostname: This is how you set the computer name. Our current convention is to use lowercase plankline followed by a number: e.g. **plankline-7**

```
hostnamectl set-hostname plankline-1
```

Add users:

```
sudo adduser XXXX
sudo usermod -aG sudo XXXXX
sudo usermod -aG plankline XXXXX
```

1.2.3 3. Misc Setup

Setup a scratch filesystem: To setup a very fast scratch drive, add the following mount configuration to `/etc/fstab`

```
tmpfs /tmp tmpfs defaults,noatime,nosuid,nodev,mode=777,size=60G 0 0
```

Setup Automounting: On Ubuntu, automounting is configured by the fstab file and is automatically run during system startup.

```
lsblk
sudo nano /etc/fstab
```

"/dev/sdX /media/plankline/Data ext4 defaults,rw,user 0 0"

```
mkdir -p /media/plankline/data
sudo chmod -R 777 /media/plankline
sudo mount -a
```

Set swapiness: This is a minor performance tweak to diswade the computer from ever trying to use hard drive swap space over system memory. These computer generally have plenty of memory and have no need for offloading to physical media.

```
sudo nano /etc/sysctl.conf
```

add "vm.swappiness=1" to the file.

1.2.4 4. Setup Backups

Timeshift is a useful tool for maintaining backups of linux directories. Only needs to be done once per computer.

```
sudo apt-get install timeshift
sudo timeshift --create
```

Using NoMachine (or local monitor), open timshift and setup automatic scheduled backups. I'd recommend keeping 5 daily, 4 weekly, and 12 monthly backups. Or, using the terminal:

```
sudo nano /etc/timeshift/timeshift.json
```

1.2.5 5. Networking

Setup Firewall: To check the status of the firewall:

```
sudo ufw status
```

To allow connections on particular ports (e.g. 80 for *Shiny* and 19999 for *netdata*):

```
sudo ufw allow 8000:8100/tcp
sudo ufw allow 19999
sudo ufw allow 9993
sudo ufw allow http
sudo ufw limit ssh
sudo ufw allow "CUPS"
sudo ufw allow "Samba"
sudo ufw allow 4000/tcp
sudo ufw allow 4080
sudo ufw allow 4443
sudo ufw allow 4011:4999/udp
```

To turn off the firewall:

```
sudo ufw disable
```

To View Rules and Delete them:

```
sudo ufw status numbered
sudo ufw delete XX
```

Install intel 10G driver: Download from Intel X520 IXGBE driver from a trusted source. *TODO: include a copy of the driver on a shared networked location.*

```
tar xzf ./ixgbe-5.16.5.tar.gz
cd ./ixgbe-5.16.5/src/
sudo make install
sudo nano /etc/default/grub
```

Add "ixgbe.allow_unsupported_sfp=1" to GRUB_CMDLINE_LINUX="" line

```
sudo grub-mkconfig -o /boot/grub/grub.cfg
sudo rmmod ixgbe && sudo modprobe ixgbe allow_unsupported_dfp=1
sudo update-initramfs -u
```

Check to see if network interface is present, e.g. eno1

```
ip a
```

Additional network inspection can be performed by ethtool:

```
ethtool <interface>
ethtool eno1
```

Setup NFS: (Linux and Windows networking shares) *work in progress*

_On a windows computer, you need to install the NFS support as an optional windows add-on. In the start menu > Turn Windows Features On and Off > NFS Support. Install NFS client. _

```
sudo apt install nfs-kernel-server
sudo systemctl start nfs-kernel-server.service

nano /etc/exports
```

Add the following lines:

```
/media/plankline/ingest    *(rw,sync,subtree_check)
```

Setup Samba: (Windows SMB)

```
sudo apt-get install samba samba-client
sudo nano /etc/samba/smb.conf
```

Inside the smb.conf file, go to the bottom of the file and add the following (or something similar). This will setup a share named *ingest* which will be writable by everyone (no security) on the network.

```
[ingest]
    comment = Plankline data drive.
    path = /media/plankline/ingest
    browsable = yes
    guest ok = yes
    read only = no
    create mask = 0777
```

Can do the same for an output directory share.

```
[output]
    comment = Plankline output drive.
    path = /media/plankline/data
    browsable = yes
    guest ok = yes
    read only = no
    create mask = 0777
```

Create the directory given above:

```
sudo mkdir -p /media/plankline/ingest
sudo chown nobody:nogroup /media/plankline/ingest
```

Restart samba to apply the edits.

```
sudo systemctl restart smbd.service
```

The folder /media/plankline/ingest should now be available as \plankline-X\ingest from any file manager or as http://plankline-x/ingest from the browser.

FTP:

Mount NAS on Plankline:

```
sudo apt-get install cifs-utils
sudo mkdir /mnt/shuttle
sudo chmod 777 /mnt/shuttle
sudo chown nobody:nogroup /mnt/shuttle
sudo mount -t cifs -o user=tbkelly //10.25.187.104/shuttle /mnt/shuttle
sudo nano /etc/fstab
```

Add line: "//10.25.187.104/shuttle /mnt/shuttle cifs user=<user>,pass=<password> 0 0"

1.2.6 6. Misc Linux Tasks

Install NoMachine:

https://www.nomachine.com/download

RAID-based data drive:

After setting up the disk stucture in the BIOS, open *disks* to create a new partition on the array. Choose XFS file system (or EXT4).

1.3 Performance Testing

1.3.1 Disk speed

To test disk IO, here's a one liner that will write a file of all zero's to a target location.

```
dd if=/dev/zero of=<testing dir>/tmp.data bs=10G count=1 oflag=dsync
```

Example Results for *Plankline-2*:

```
plankline-2:/media/plankline/Data/Data      654 MB/s
plankline-2:/tmp                            1100 MB/s
plankline-2:/home/tbkelly                   928 MB/s
```

Recommended test set:

```
dd if=/dev/zero of=/media/plankline/Data/Data/tmp.data bs=10G count=1 oflag=dsync
dd if=/dev/zero of=/tmp/tmp.data bs=10G count=1 oflag=dsync
dd if=/dev/zero of=/home/tbkelly/tmp.data bs=10G count=1 oflag=dsync
```

Chapter 2

Notes on the installation and use of morphocluster at UAF

2.1 Installation

Generally just follow the outline provided here: [https://github.com/morphocluster/morphocluster]

Install docker if haven't already:

```
sudo aptitude install docker docker-compose npm dos2unix

cd /opt
git clone https://github.com/tbrycekelly/UAF-Morphocluster.git
mkdir -p /media/plankline/Data/Morphocluster/data
mkdir -p /media/plankline/Data/morphocluster/postgres
```

Add users to the docker group so you can control the docker installations.

```
sudo usermod -aG docker USERNAME
```

Need to fix some possible issues:

```
sudo nano /etc/environment
```

Add lines: COMPOSE_DOCKER_CLI_BUILD=1 DOCKER_BUILDKIT=1

Save. Ctrl + x y. Then build the container. This step will take a while the first time (will be shorter if you are rebuilding the image later on).

```
cd /opt/UAF-Morphocluster/morphocluster
sudo docker-compose up --build
```

From a new terminal, move to the contianer directory and run

```
cd /opt/UAF-Morphocluster/morphocluster
sudo docker-compose exec morphocluster bash
```

Then within the container:

2.2 Activate conda environment

(base) root@abc123...:/code# . ./activate (morphocluster) root@abc123...:/code#

2.3 Create a user account

flask add-user test-user Adding user test-user: Password: <hidden> Retype Password: <hidden>

2.3.1 Running

2.3.2 Troubleshooting and Misc

2.3.2.1 Docker hints

General info about the running containers

```
docker ps
docker stats
```

Container-specific information

```
docker logs morphocluster
docker top morphocluster
docker port morphocluster
```

Container-specific actions

```
docker run morphocluster
docker stop morphoclsuter
docker kill morphocluster
docker restart morphocluster
```

Chapter 3

Plankline Setup

This is an *actively* developed set of documentation for the setup and use of the UAF version of the planktline processing pipeline developed by OSU. Please contact Thomas Kelly tbkelly@alaska.edu for more information.

Updated 2023-08-11

3.0.1 1. Segmentation

On the first install,

```
cd /opt
sudo git clone https://github.com/tbrycekelly/UAF-Segmentation
sudo chown -R plankline:plankline /opt/UAF-Segmentation
sudo chmod 776 -R /opt/UAF-Segmentation
```

To update the executable with a new version:

```
cd /opt/UAF-Segmentation
git clean -f
git pull
```

To build the segmentation executable:

```
cd /opt/UAF-Segmentation/build
cmake ../
make
```

Test it and copy it to final directory (if it works):

```
./segment
cp ./segment ..
```

3.0.2 2. SCNN

For the first install,

```
cd /opt
sudo git clone https://github.com/tbrycekelly/UAF-SCNN.git
sudo chown -R plankline:plankline /opt/UAF-SCNN
sudo chmod 776 -R /opt/UAF-SCNN
```

To update from github,

```
cd /opt/UAF-SCNN
git clean -f
git pull
```

May need to run `git config --global --add safe.directory /opt/UAF-SCNN` if permissions are not right.

To build SCNN:

```
cd /opt/UAF-SCNN/build
make clean
make wp2
```

If there are undefined refences on the make, check that the LIBS line in ./build/Makefile is the output of `pkg-config opencv --cflags --libs` and inclues `-lcublas`.

Test it and copy it to final directory (if it works):

```
./wp2
cp ./wp2 ../scnn
```

3.0.2.1 2.1 Training a Neural Network

Copy the training dataset into /opt/UAF-SCNN/Data/plankton/train so that images are in subfolders by category, e.g.: /opt/UAF-SCNN/Data/plankton/train/detritus/iamge.jpg

Run classList.sh

```
cd /opt/UAF-SCNN/Data/plankton
./classList.sh
```

You may wish to change the minimum sample size required for a taxa to be included by modifying the *minN* value within classList.sh. Taxa folders with fewer than *minN* images will not be included in the training.

TODO: Complete this section

3.0.3 3. Plankline Scripts

This can be performed by any/every user on a system and can be placed in any folder. So far we have been placing this folder, /UAF-Plankline/ in Tom's documents folder. Typical locations that make sense include ∼ (home directory), ∼/Desktop, ∼/Documents.

```
cd ~
git clone https://github.com/tbrycekelly/UAF-Plankline.git
```

To update an existing folder (will remove the config files present):

```
cd ~/UAF-Plankline
git clean -f
git pull
```

To run plankline with a specific configuration file (required):

```
python3 segmentation.py -c <ini> -d <dir>
python3 classification.py -c <ini> -d <dir>
```

So for example:

```
python3 segmentation.py -c osu_test_config.ini -d /media/plankline/data/test_data
```

The segmentation.py script will read in a project directory containing a *raw* subfolder of AVIs and will create a *segmentation* subfolder if not already existing. Each AVI will be processed, flatfielded, and cropped to a dedicated TAR file inside *segmentation*. If using the optional *compress = True* flag then the file will be a TAR.GZ file. Simiarly the *classification.py* script will read in a project directory and for every file in *segmentation* produce a classification results file in a *classification* subfolder. Both scripts will place a copy of the configuration file used into their respective subfolders for archival purposes.

These scripts require the segmentation executable and SCNN executable to be available (*see 1 and 2*).

Chapter 4

Documentation for UAF-Plankline

N.B. This version of plankline was forked and adapted significantly from the original OSU version(DOI: 10.5281/zenodo.4641158) [1]. The following is a high-level summary of the changes applied in no particular order.

1. The segmentation and SCNN executables have been relocated to /opt/Threshold-MSER and /opt/SCNN, respectively. This allows all users to use these utilities without having to change/fix the paths.
2. All settings for segmentation and classification are now set within the .ini file itself. This is the only file that should be modified between projects/runs/etc. I am still working on the inline documentation in the config file. Eventually will make a "default.ini" to make copies of.
3. A new set of logging utilities are built in to allow for detailed and configurable logging routines to be used. All log entries are timestamped and given a status level: INFO, DEBUG, WARN, ERROR, CRITICAL. Default log files are places into the working directory (e.g. /media/plankline/Data/Data/projectX/) and timestamped. Logs for the executables are located in their respective folders.
4. Only some messages are printed to the screen by default (except warnings and errors), including only key information. A progress bar is shown for the majority of the processing.
5. Both scripts are now run by calling the python file directly.
6. Currently I have not implemented/adapted the previous directory/file verification functions that plankline used. So both scripts will rewrite all existing files and does not check/verify if output exists.

4.1 Installation

Please see /home/thomas/Github/UAF-Plankline/Notes/Plankline Setup.md "Plankline Setup" for installation instructions.

4.2 Segmentation (MSER)

The segmentation algorithm used in the *Plankline* processing suite is nomally a **Maximally Stable Extremal Region** approach as implemented in *opencv*, but there are caveats. Depending onthe signal to noise ratio hyperparamter (**SNR**), one of three possible segmentation routines are called. SNR is calculated as follows (imageProcessing.↩ cpp:497).

```
float SNR(const cv::Mat& img) {
    // perform histogram equalization
    cv::Mat imgHeq;
    cv::equalizeHist(img, imgHeq);

    // Calculate Signal To Noise Ratio (SNR)
    cv::Mat imgClean, imgNoise;
```

```
    cv::medianBlur(imgHeq, imgClean, 3);
    imgNoise = imgHeq - imgClean;
    double SNR = 20*( cv::log(cv::norm(imgClean,cv::NORM_L2) / cv::norm(imgNoise,cv::NORM_L2)) );

    return SNR;
```

When the SNR of an image is greater than that provided, then the MSER algorithm is called immediately after flatfielding and preprocessing (with a `3x3` kernel) [2]. If the SNR is greater than 75% of the hyperparamter value, then the image is flatfielded, proprocessed (with a `17x17` kernel), and then thresholded based on the **threshold** hyperparameter. A contouring algorithm is then applied to find specific ROIs. Finally, if the image SNR is below 75% of the hyperparamter value, then the image is flatfielded, proprocessed using the same `17x17` kernel and otherwise processed identically to the previous processing (but in a standalone function).

4.3 Training

4.4 Classification

4.4.1 References

[1] Schmid, Moritz S, Daprano, Dominic, Jacobson, Kyler M, Sullivan, Christopher, Briseño-Avena, Christian, Luo, Jessica Y, & Cowen, Robert K. (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 (1.0.0). Zenodo. https://doi.org/10.5281/zenodo.4641158

[2] Preprocessing here entails an erosion and dialation step conducted by *opencv*. The erosion kernel size is `2*size+1` and thus is a `3x3` or `17x17`

Chapter 5

Plankline Scripts

Scripts were originally from the Plankline project controlled by OSU (citation's to be included). I have heavily modified and repurposed the code to suit our architecture and to improve logging and performance. These scripts are available as is and I make no claim to ownership or control of any potentially sensitive code. These are intended for in-house use at UAF.

5.1 Setup and Documentation

The Plankline suite of processing scripts is split into three main modules: (1) segmentation, (2) classification, (3) analysis; and is unified with the scripts in this repository.
If you are new to linux and setting up a Plankline instance for the first time, please start with the Linues Setup and the Notes/General%20Setup.md "Plankline Setup" guides.
For documentation on how Plankline works, please see the Plankline reference.
Optionally we also have a preliminary Morphocluster setup guide for those that are interested.

5.2 Quick start

To run plankline with a specific configuration file and input directory:

```
python3 segmentation.py -c <ini> -d <dir>
python3 classification.py -c <ini> -d <dir>
```

So for example:

```
python3 segmentation.py -c osu_test_config.ini -d /media/plankline/data/test_data
```

Chapter 6

Namespace Index

6.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 7

Namespace Documentation

7.1 classification Namespace Reference

Functions

- def classify (tar_file)

Variables

- string **v_string** = "V2023.09.05"
- **parser** = argparse.ArgumentParser(description="Classification tool for managing the isiis_scnn processes")
- **required**
- **True**
- **help**
- **args** = parser.parse_args()
- **config** = configparser.ConfigParser()
- **working_dir** = os.path.abspath(args.directory)
- **defaults**
- **logger** = logging.getLogger('sLogger')
- **permis** = int(config['general']['dir_permissions'])
- **scnn_instances** = int(config['classification']['scnn_instances'])
- **scnn_directory** = config['classification']['scnn_dir']
- **scnn_command** = config['classification']['scnn_cmd']
- **epoch** = int(config['classification']['epoch'])
- **fast_scratch** = config['segmentation']['fast_scratch']
- string **segmentation_dir** = working_dir + "/segmentation"
- string **classification_dir** = working_dir + "/classification"
- **exist_ok**
- list **tars** = [os.path.join(segmentation_dir, tar) for tar in os.listdir(segmentation_dir) if tar.endswith(".tar.gz")]
- **num_gpus** = str(subprocess.check_output(["nvidia-smi", "-L"])).count('UUID')
- **queue** = Queue()
- **num_processes** = scnn_instances ∗ num_gpus
- **tar_length** = len(tars)
- **p** = Pool(num_processes)
- **timer_pool** = time()
- **total**
- **ignore_errors**
- string **cp_file** = classification_dir + '/' + str(datetime.datetime.now()) + ' ' + args.config

7.1.1 Detailed Description

```
Classification script for UAF-Plankline

Usage:
    ./classification.py -c <config.ini> -d <project directory>

License:
    MIT License

    Copyright (c) 2023 Thomas Kelly

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.
```

7.1.2 Function Documentation

7.1.2.1 classify()

```
def classification.classify (
              tar_file )

Classify images contained within a TAR file.
```

Definition at line 47 of file classification.py.

```
47 def classify(tar_file):
48     """Classify images contained within a TAR file."""
49
50     timer_classify = time()
51     logger.info("Starting classify")
52     date = datetime.datetime.now().strftime("%Y-%m-%d")
53     basename = config['classification']['basename']
54     logger.debug(f"Basename for model run is {basename}.")
55     logger.info(f"Current ram usage (GB): {psutil.virtual_memory()[3]/1000000000:.2f}")
56     logger.info(f"Current cpu usage (%): {psutil.cpu_percent(4):.1f}")
57
58     if config['general']['compress_output'] == 'True':
59         image_dir = tar_file.replace(".tar.gz", "") # remove extension
60         tar_identifier = os.path.basename(image_dir)
61         tmp_dir = fast_scratch + '/' + tar_identifier
62         os.makedirs(tmp_dir, permis, exist_ok = True)
63         log_file = f"{classification_dir}/{tar_identifier}-{date}.log"
64     else:
65         image_dir = tar_file.replace(".tar", "") # remove extension
66         tar_identifier = os.path.basename(image_dir)
67         tmp_dir = fast_scratch + '/' + tar_identifier
68         os.makedirs(tmp_dir, permis, exist_ok = True)
69         log_file = f"{classification_dir}/{tar_identifier}-{date}.log"
70
71     image_dir = tmp_dir
72
73     gpu_id = queue.get()
74
75     logger.info(f"Starting on GPU {gpu_id}")
76     logger.info(f'image_dir: {image_dir}')
77     logger.info(f'tar_identifier: {tar_identifier}')
78
79     # Untar files
80     if config['general']['compress_output'] == 'True':
```

```
81          untar_cmd = f'tar -xzf "{tar_file}" -C "{image_dir}" --strip-components=4 --wildcards "*.png"  »
    "{log_file}" 2>&1' # TBK change strip-components to what you need.
82          logger.debug('Untarring+unzipping files: ' + untar_cmd)
83      else:
84          untar_cmd = f'tar -xf "{tar_file}" -C "{image_dir}" --strip-components=4 --wildcards "*.png"  »
    "{log_file}" 2>&1' # TBK change strip-components to what you need.
85          logger.debug('Untarring files: ' + untar_cmd)
86
87      timer_untar = time()
88      os.system(untar_cmd)
89      timer_untar = time() - timer_untar
90      logger.debug(f"Untarring files took {timer_untar:.3f} s.")
91
92      # Perform classification.
93      scnn_cmd  = f"cd '{scnn_directory}'; nohup '{scnn_command}' -start {epoch} -stop {epoch} -unl
    '{image_dir}' -cD {gpu_id} -basename {basename} » '{log_file}' 2>&1"
94      logger.debug('Running SCNN: ' + scnn_cmd)
95      logger.info('Start SCNN.')
96
97      timer_scnn = time()
98      os.system(scnn_cmd)
99      timer_scnn = time() - timer_scnn
100      logger.info('End SCNN.')
101      logger.debug(f"SCNN took {timer_scnn:.3f} s.")
102
103      # Move the csv file resulting from classification.
104      logger.info(f"Looking for files in {scnn_directory}/Data/{basename}/ that match the id:
    {tar_identifier}")
105      csv_path = glob.glob(f"{scnn_directory}/Data/{basename}/*{tar_identifier[10:]}*")
106      if len(csv_path) > 0:
107          csv_path = csv_path[0]
108          csv_file = f"{classification_dir}/{tar_identifier}.csv"
109
110          timer_move = time()
111          shutil.move(csv_path, csv_file)
112          os.chmod(csv_file, permis)
113          timer_move = time() - timer_move
114          logger.debug(f"Moving csv(s) took {timer_move:.3f} s.")
115
116      else:
117          logger.error(f'No classification file found for {tar_identifier}')
118
119      # Clean directories to make space.
120      shutil.rmtree(image_dir)
121
122      queue.put(gpu_id) # add the gpu id to the queue so that it can be allocated again
123      logger.debug('End classify.')
124      timer_classify = time() - timer_classify
125      logger.debug(f"Total classification process took {timer_classify:.3f} s.")
126
127      if config['R']['preprocess']:
128          timer_pre = time()
129          logger.debug('Preprocessing requested.')
130
131          pre_cmd = 'Rscript "' + config['R']['script'] + '" ' + config['R']['dt'] + ' ' +
    config['R']['p_threshold'] + ' "' + csv_file + '"'
132          logger.debug(f"Running preprocessing cmd: {pre_cmd}")
133          os.system(pre_cmd)
134
135          timer_pre = time() - timer_pre
136          logger.debug(f"Preprocessing took {timer_pre:.3f} s.")
137
138
```

7.2 cleanup Namespace Reference

Functions

- def **get_size** (start_path='.')

Variables

- string **v_string** = "V2023.02.02"
- **parser** = argparse.ArgumentParser(description="")
- **required**
- **True**
- **help**
- **action**

- **args** = parser.parse_args()
- working_dir = os.path.abspath(args.directory)

  Read in config options:
- **ignore_errors**

7.2.1 Detailed Description

```
Cleanup script for UAF-Plankline

Usage:
    ./cleanup.py -c <config.ini> -d <project directory>

License:
    MIT License

    Copyright (c) 2023 Thomas Kelly

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.
```

7.3 pull_all Namespace Reference

Functions

- def **directory** (arg)
- def **csv_type** (arg)
- def **float01** (arg)
- def **add_unique_postfix** (fn)
- def **get_parser** ()
- def **validate_args** (parser)
- def **find_all_taxon** (csv)
- def **invalid_taxon** (parser, taxon_exist, all_taxon)
- def **print_taxon** (taxon_locations)
- def **find_top** (csv, taxon_locations)
- def **find_above_prob** (csv, taxon_locations, probability)
- def **untar** (tar, out_dir)
- def **build_structure** (path_array, taxon_locations)

Variables

- string **v_string** = "V2023.09.05"
- def **parser** = get_parser()
- **csvs**
- **tars_dir**
- **out_dir**
- **file_dict**
- **min_images**

- **best**
- **probability**
- **different**
- **strict**
- **f** = open(csvs[0])
- def **e_taxon** = find_all_taxon(_csv)
- **name_line** = next(_csv)
- **tar_file** = file_dict[csv_file]
- **matched_images** = dict()
- **path_array** = matched_images[os.path.basename(img)]

7.3.1 Detailed Description

```
Image Pulling script for UAF-Plankline

Usage:
    ./pull_all.py -d <project directory>

License:
    MIT License

    Copyright (c) 2023 Thomas Kelly

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.
```

7.4 segmentation Namespace Reference

Functions

- def seg_ff (avi, seg_output, SNR, segment_path)
- def local_main (avi)
- def FixAviNames (avis)

Variables

- string **v_string** = "V2023.09.05"
- **parser** = argparse.ArgumentParser(description="Segmentation tool for the plankton pipeline. Uses ffmpeg and seg_ff to segment a video into crops of plankton")
- **required**
- **True**
- **help**
- **args** = parser.parse_args()
- **config** = configparser.ConfigParser()
- **working_dir** = os.path.abspath(args.directory)
- defaults

  Setup logger.

- **logger** = logging.getLogger('sLogger')
- string **cp_file** = working_dir + '/' + str(datetime.datetime.now()) + ' ' + args.config
- permis = int(config['general']['dir_permissions'])

  Read in config options:
- **SNR** = int(config['segmentation']['signal_to_noise'])
- **num_processes** = int(config['segmentation']['segment_processes'])
- **segment_path** = config['segmentation']['segment']
- **fast_scratch** = config['segmentation']['fast_scratch']
- string **raw_dir** = working_dir + "/raw"
- string **segment_dir** = working_dir + "/segmentation"
- **exist_ok**
- list **avis** = [ ]
- **p** = Pool(num_processes)
- **timer_pool** = time()
- **total**
- **ignore_errors**

7.4.1 Detailed Description

```
Segmentation script for UAF-Plankline

Usage:
    ./segmentation.py -c <config.ini> -d <project directory>

License:
    MIT License

    Copyright (c) 2023 Thomas Kelly

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.
```

7.4.2 Function Documentation

7.4.2.1 FixAviNames()

```
def segmentation.FixAviNames (
              avis )

 Helper function to remove spaces in names.
TODO: I think this is unnecessary any more.
```

Definition at line 134 of file segmentation.py.

```
134 def FixAviNames(avis):
135     """ Helper function to remove spaces in names.
136     TODO: I think this is unnecessary any more.
137     """
138     n = 0
139     for Name in avis:
```

```
140         FixedName = Name.replace(' ', '-')
141         if not (FixedName == Name):
142             os.rename(Name, FixedName)
143         avis[n] = FixedName
144         n += 1
145     return avis
146
147
```

7.4.2.2 local_main()

```
def segmentation.local_main (
               avi )
```

A single threaded function that takes one avi path.

Definition at line 69 of file segmentation.py.

```
69 def local_main(avi):
70     """A single threaded function that takes one avi path."""
71     logger.info("Starting local_main.")
72
73     # setup all of the local paths for the avi
74     avi_file = os.path.basename(avi) # get only the file part of the full path
75     avi_date_code = os.path.splitext(avi_file)[0] # remove .avi
76     avi_segment_scratch = fast_scratch + "/" + avi_date_code
77     seg_output = avi_segment_scratch + "_s"
78     out_dir = segment_dir + "/" + avi_date_code
79
80     logger.info(f'avi_file: {avi_file}')
81     logger.info(f'avi_date_code: {avi_date_code}')
82     logger.info(f'avi_segment_scratch: {avi_segment_scratch}')
83     logger.info(f'seg_output: {seg_output}')
84     logger.info(f'segment_dir: {segment_dir}')
85     logger.info(f'working_dir: {working_dir}')
86     logger.info(f"Current ram usage (GB): {psutil.virtual_memory()[3]/1000000000:.2f}")
87     logger.info(f"Current cpu usage (%): {psutil.cpu_percent(4):.1f}")
88
89
90     logger.debug(f'Starting AVI file: {avi_file}')
91
92     # Create necessary directory structure.
93     logger.info('Setting up AVI directories.')
94     os.makedirs(avi_segment_scratch, permis, exist_ok=True)
95     os.makedirs(seg_output, permis, exist_ok=True)
96
97     # Segmentation.
98     logger.info('Starting segmentation.')
99     timer_seg = time()
100     seg_ff(avi, seg_output, SNR, segment_path)
101     timer_seg = time() - timer_seg
102     logger.debug(f"Segmentation executable took {timer_seg:.3f} s.")
103
104     if config['general']['compress_output'] == 'True':
105         logger.info('Start tarring+compressing.')
106         tar_name = out_dir + ".tar.gz"
107         tar = f'tar czf \"{tar_name}\" -C \"{seg_output}\" .'
108         logger.debug(tar)
109
110         timer_tar = time()
111         os.system(tar)
112         os.chmod(tar_name, permis)
113         timer_tar = time() - timer_tar
114
115         logger.info(f'End tarring+compressing in {timer_tar:.3f} s.')
116     else:
117         logger.info('Start tarring')
118         tar_name = out_dir + ".tar"
119         tar = f'tar cf \"{tar_name}\" -C \"{seg_output}\" .'
120         logger.debug(tar)
121
122         timer_tar = time()
123         os.system(tar)
124         os.chmod(tar_name, permis)
125         timer_tar = time() - timer_tar
126
127         logger.info(f'End tarring in {timer_tar:.3f} s.')
128
129     shutil.rmtree(seg_output)          # remove datecode_s/
130     shutil.rmtree(avi_segment_scratch) # remove datecode/
131     logger.info('End local_main.')
132
133
```

7.4.2.3 seg_ff()

```
def segmentation.seg_ff (
              avi,
              seg_output,
              SNR,
              segment_path )

Formats and calls the segmentation executable
```

Definition at line 46 of file segmentation.py.

```
46 def seg_ff(avi, seg_output, SNR, segment_path):
47     """Formats and calls the segmentation executable"""
48     snr = str(SNR)
49     epsilon = config['segmentation']['overlap']
50     delta = config['segmentation']['delta']
51     max_area = config['segmentation']['max_area']
52     min_area = config['segmentation']['min_area']
53 
54     segment_log = working_dir + '/segmentation/segment_' + str(datetime.datetime.now()) + '.log'
55     full_output = config['segmentation']['full_output']
56 
57     seg = f'nohup \"{segment_path}\" -i \"{avi}\" -o \"{seg_output}\" -s {snr} -e {epsilon} -M {max_area}
       -m {min_area} -d {delta} {full_output} » \"{segment_log}\" 2>&1'
58     logger.info("Segmentation call: " + seg)
59 
60     os.chmod(seg_output, permis)
61 
62     timer_seg = time()
63     os.system(seg)
64     timer_seg = time() - timer_seg
65     logger.debug(f"Segmentation finished in {timer_seg:.3f} s.")
66 
67 
68 
```

7.5 train Namespace Reference

Variables

- string **v_string** = "V2023.09.05"
- **parser** = argparse.ArgumentParser(description="")
- **required**
- **True**
- **help**
- **args** = parser.parse_args()
- **config** = configparser.ConfigParser()
- **config_version** = config['general']['config_version']
- **model_dir** = config['training']['model_dir']
- fast_scratch = config['training']['fast_scratch']

  Setup scratch for training.
- **scnn_cmd** = config['training']['scnn_cmd']
- **start** = config['training']['start']
- **stop** = config['training']['stop']
- **batchsize** = config['training']['batchsize']
- **basename** = config['training']['basename']
- **vsp** = config['training']['validationSetRatio']
- **lrd** = config['training']['learningRateDecay']
- **ilr** = config['training']['initialLearningRate']
- **permis** = int(config['general']['dir_permissions'])
- defaults

  Setup logger.
- **logger** = logging.getLogger('sLogger')
- string **fast_data** = fast_scratch + "/Data"
- string **fast_weights** = fast_scratch + "/weights"

- string **cp_file** = model_dir + '/' + str(datetime.datetime.now()) + ' ' + args.config
- **time_copy** = time()
- **timer_train** = time()
- string train = f\"{scnn_cmd}\" -project {fast_scratch} -start {i} -stop {i+1} -batchSize {batchsize} -basename {basename} -vsp {vsp} -lrd {lrd} -ilr {ilr} -cD 0'

  Format training call:
- list **train_call** = [scnn_cmd, '-project', fast_scratch, '-start', str(i), '-stop', str(i+1), '-batchSize', batchsize, '-basename', basename, '-vsp', vsp, '-lrd', lrd, '-ilr', ilr, '-cD', '0']
- **result** = subprocess.run(train_call, stdout = subprocess.PIPE)
- **sep**
- **ignore_errors**

7.5.1 Detailed Description

```
Training script for UAF-Plankline
    This is the training script used to facilitate training of new SCNN models.
    Settings for this script come exclusively from the configuration ini file
    passed:

        e.g. python3 train.py -c config.ini

    Importantly, teh script copies all data to a temporary scratch directory and
    then copies results back onces completed. If there is a failure then no model
    epochs will be saved. The user is free to grab them from the scratch_dir.

Usage:
    ./train.py -c <config.ini>

License:
    MIT License

    Copyright (c) 2023 Thomas Kelly

    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files (the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
    SOFTWARE.
```

Index