Scott Base Seal Monitoring Application User Manual

Authored by Jacques Terblanche

Introduction

This document is intended to aid users who may be unfamiliar with the application with using it or making modifications. This project does not have a user interface, so most interaction is done directly through an IDE using a Jupyter Notebook or running scripts through the command line.

This application is intended to be used for images captured of the Weddell Seal colony at Scott Base, particularly using a camera located at Crater Hill. Datasets of similar image quality of Antarctic seal colonies may also work, but effectiveness is not guaranteed.

First-Time Setup

This deployment has only been tested on Linux machines and may not work well on other operating systems.

1. Clone the repository. Navigate to the folder you want to clone the repository into and use this command:

```
git clone https://github.com/qBLANCK/Scott-Base-Seal-Monitoring.git
```

2. Set up environment. Install all of the dependencies required for this project by running these commands.

```
/install_conda.sh
conda env create -f environment.yaml
```

3. Activate the environment.

```
conda activate seal_env
```

The application should now be ready to use. Step 3 is the only one that will need to be repeated. Each time you want to run one of the following scripts make sure that the environment is activated.

Processing a Dataset

1. Cropping and Converting the Dataset

Runtime: Approx. 4 hours per 1000 images

The dataset must first be cropped and converted to a format that can be processed by the application. This involves removing unnecessary space from the images and converting the images from Panasonic Raw format (.RW2) to JPEG files. If your dataset is already in JPEG format, this can be skipped, however, you may still want to consider cropping your images.

A ‘crop box’ needs to be defined which will be the area of the image that is kept. This takes four numbers - the x and y coordinates of the upper left corner and the lower right corner. For example, the 2021-22 dataset uses a crop box with values `4600 7250 8514 7625` and the 2022-23 dataset uses `6000 6900 15750 7800`
Once you have defined your cropped region, you can run the processing script using this command.

```
python -m scripts.crop_and_convert --input "/path/to/your/rw2/files"
--output "/path/to/your/processed/images" --crop_box 1 2 3 4
```

The processed images should look something like this:

All of the processed images will be named according to the date and time they were taken in YYYY-MM-DDTHH_MM_SS format, e.g., 2021-11-20T10_02_26.jpg

You can also include an option to scale the image after it has been cropped by including `--scale` in the command. More information on each parameter can be found by using

```
python -m scripts.crop_and_convert --help
```

2. Generating Seal Locations

Runtime: Approx. 1-2 hours

Once you have your cropped dataset, you can use a supplied model to locate seals in the image. If you have a new dataset you will most likely want to create a ‘mask’ image to define any regions you know seals won’t be detected in, such as the rocky area around Scott Base. Your mask image should have only black and white pixels, where white pixels represent the area you want to *exclude.* This can be achieved using something like Microsoft Paint, but if you have access to tools like Photoshop or GIMP, this will make it easier. Your mask should look something like this:

Make sure that your mask image is the *exact same size* as the images in your dataset. Be careful not to compress the image when creating it.

Once you have created your mask, you can process your dataset using this command:

```
python -m scripts.locate_seals --input "/path/to/your/processed/images"
--mask "/path/to/your/mask/image.jpg" --output
"/path/to/your/output/file.csv"
```

There are a few other parameters you can also specify:

- `--model` The model used to detect seals. The default model supplied is the best one generated through this project.
- `--confidence` The confidence threshold to detect seals at. Any detections below this threshold will not be added to the output.
- `--brightness` The brightness threshold to brighten dark images. If you do not want to use a brightness threshold, set this to 0.

This will output a spreadsheet in CSV format containing every seal detection the model made. E.g.,

```
Timestamp,X_min,Y_min,X_max,Y_max,Confidence,Timelapse_pos
2022-10-19T16_59_31,5067,795,5080,809,0.841,0
2022-10-19T16_59_31,9045,723,9071,749,0.771,0
2022-10-19T16_59_31,7435,838,7448,851,0.721,0
```

3. Filtering Seal Locations (Optional)

Runtime: Less than 5 minutes

Filtering is done based on adjacent images. For every seal detection, neighbouring images are checked to see if a matching detection can be found nearby. This is useful for reducing false positives, especially in outlier images with a large number of false detections, however, it will also likely affect some real seals.
This requires a CSV file in the format created by the script for locating seals.

```
python -m scripts.filter_seal_locations --input
"/path/to/your/locations/file.csv" --output
"/path/to/your/filtered/locations/file.csv"
```

Again, additional parameters can be specified:

- `--distance` The distance in which a neighbouring detection must be found to validate a detection. The default distance is 25. Higher values may let too many false positives slip through, but too low will also filter out true seals.

- `--confidence` The confidence threshold to filter. Any detections below this threshold will be filtered out.
- `--num_timestamps` The number of images to check on either side of each image. Must be at least 1.
- `--num_to_verify` The number of times a detection needs to be matched in neighbouring images to be validated.

The format of the filtered CSV file will be the same as the input file.

4. Generating Seal Counts

Runtime: Less than 1 minute

Now that you have your CSV file of locations, you can count up the locations at each timestamp, or you can get this script to do it for you.

```
python -m scripts.counts_from_locations --input
"/path/to/your/locations/file.csv" --output
"/path/to/your/counts/file.csv"
```

This will generate a CSV file in a format like this:

```
Timestamp,Count (t=30),Count (t=40),Count (t=50)
2021-11-20T10_02_26,22,22,16
2021-11-20T10_17_26,22,21,16
2021-11-20T10_32_26,24,20,17
```

You can specify which confidence thresholds are included in the model by using the `--thresholds` parameter. By default, these are `30 40 50 60 70`.

5. Add Snowstorm Detection

Runtime: Approx. 25-35 minutes

This augments the seal counts CSV file with snowstorm detections so that false low counts of seals can be marked. This requires another run through the image directory to calculate the average brightness so high-brightness images can be used to detect snowstorms. It also uses a mask similar to the one used for Step 2. You can probably re-use that mask, but for best results, you could also create a new one that covers the areas of your dataset that are usually darkest (e.g. rocky areas).

```
python -m scripts.detect_snowstorms --input_dir
"/path/to/your/processed/images" --input_csv
"/path/to/your/counts/file.csv" --mask "/path/to/your/mask/image.jpg"
--output "/path/to/your/output/file.csv"
```

You can specify specific brightness thresholds for snowstorms and the confidence interval of the counts you want to use using `--brightness` and `--confidence`.

6. Generate Timelapse

Runtime: Approx 15-30 minutes

To generate a timelapse video of your dataset you can use the command

```
python -m scripts.heatmap.create_timelapse --input
"/path/to/your/processed/images" --output "/path/to/your/timelapse.mp4"
```

You can also include the `--scale` parameter to increase or decrease the dimensions of the timelapse video. This may be necessary for datasets that have particularly large images.

7. Generate Heatmap

Runtime: Approx. 20 minutes to generate each minute of timelapse
WARNING: Generating the heatmap is incredibly computationally expensive. You may need to increase the chunk size significantly to get it to work.

The heatmap will be generated in chunks. Unless you have access to a supercomputer you will almost certainly need more than one chunk. These will be stored in the folder scripts/heatmap/heatmap_chunks and named according to the order they were generated. You will need to place your timelapse video file generated in step 5 and your seal locations CSV file generated in step 3 (or 2 if you skipped filtering). Place both of these files in the scripts/heatmap folder. You will also need to know how many images are in your dataset so that the length of each chunk can be calculated correctly. Once all of this has been done you can run the script using

```
cd scripts/heatmap
python -m create_heatmap --chunks 4 --timelapse "timelapse.mp4" --seals
"locations.csv" --frames 1000
```

Modify the chunks variable to suit your needs and set the frames to the number of images in your dataset.

Once the chunks have been generated you can join them together either using video processing software or using `concat_heatmap.sh`. If you want to use this script you will need to make some modifications. The variable `num_clips` will need to be changed to the number of chunks that were generated.
By default, there is one second of redundancy in each chunk except for the first one. This overlap is to ensure that the heatmap doesn't appear to "reset" at the start of each chunk. You will need to find the length of one of the chunks (*not* the first one) and set the `length` variable in the script to this number. Then you will be able to join all of the heatmap chunks together into one file. By default, this will just be called "output.mp4"