Use Case Document: Pack-Ice Seal Survey (SBU)

Note: the red font indicates missing/partial/unclear information.

# Descriptions

* **Collaborator:** Dr. Heather Lynch, Bento Goncalves
* **Field**: Marine ecology
* **Measure of success:** Spatial coverage of surveyed area and detection accuracy
* **Primary challenges:** Need a pipeline for managing input images from large .tiff files to 299 x 299 pixel patches, and keeping track of relevant spatial metadata
* **Platforms used:**
  + **Scripting languages/library:** TensorFlow, Python, JAGS, R
  + **System:** Currently an NVIDIA GTX 1080 GPU (1 GPU with 2560 CUDA cores)
* **Physical Systems:** 
  + **System:** Sub-meter resolution commercial satellite imagery (particularly WV-3)
  + **Parameters:**
    - * + Covariates to be extracted in situ: Sea ice concentration
        + Covariates to be imported from MODIS or other auxiliary imagery: SST, chlorophyll-a, distance to coastline, distance to bathymetric feature
        + Response variable of interest: location of seals on pack-ice
* **Description:** Detection algorithm to extract the location of seals from high-resolution imagery (this is a probabilistic variable because of detection failures [false positives and false negatives])
* **Components:**
  + **Stage 1 (one time only):** a) developing customized training set for the CNN, b) estimate the correct hyperparameters, c) training the CNN
  + **Stage 2:** Input and processing of high resolution imagery
  + **Stage 3:** Input and processing of auxiliary covariate data
  + **Stage 4:** Distributing computation to GPU(s)
  + **Stage 5:** Classification of seal haul outs byTensorFlow
  + **Stage 6:** Seal counts within seal haul outs
  + **Stage 7:** Post-detection statistical modelling with JAGS and R
* **Stage 1**
  + **Description: a)** Manually inspect WV3 imagery looking for seals and other features we wish to classify. When a feature is found, extract a 450x450 pixel snippet – at true resolution -- centered on it (raw training image). Raw training images are grouped according to their intended label. Labels: 1) crabeater seal (‘crabeater’); 2) Weddell seal (‘weddell’); 3) empty pack-ice (‘pack-ice’); 4) other empty substrate (‘other’); and 5) emperor penguin (‘emperor’). Raw training images go through data augmentation, a series of transformations such as random rotations, random crops and random changes in hue, creating a larger number of ‘new’ training samples. Samples are cropped during augmentation to the desired input size of the CNN in use (299x299 for Inception Resnet V2). To ensure all classes are equally represented during training, the number of augmented images extracted from each raw image will be inversely proportional to the number of raw images in that image’s class. A small portion of training samples (~20%) will then be set apart for model validation and testing. Finally, processed training samples are compressed into ‘.tfrecord’ (TensorFlow’s native input data type) files to train the CNN.
  + **b and c)** Train the CNN on seal haul outs with TensorFlow on the training set from **a)** multiple times using distinct combinations of hyperparameters (similar to a grid search). Use the validation set to set apart combinations that obtain highest performance (i.e. accuracy, precision and recall). Save best model for next stages and record performance metrics at the test set. All training routines will be done with an adam optimizer. Hyperparameters of interest: 1) number of epochs (where 1 epoch = full run through all training samples); 2) learning rate (how fast weights get updated); 3) batch-size (how many random training images the CNN says in one iteration of the optimizer); 4 dropout-rate (portion of training samples that gets randomly discarded during training).
  + **Python files:**
* augment\_dataset.py (image augmentation routine)
* create\_tfrecord.py (compresses processed images into ‘.tfrecord’ for training)
* train\_inception.py (trains CNN on tfrecord images)
  + **Cores:** GPU enabled.
* **Input files:**
* seals\_rainining (~100MB\*, folder containing raw training images, separated in subfolders by classification label)
* seals\_validation (~10MB\*, folder containing raw validation images, separated in subfolders by classification label)
* seals\_test (~10MB\*, folder containing raw test images, separated in subfolders by classification label)
* labels.txt (text file with numbered classification labels)

\* approximate file sizes for ~ 5000 raw training images; file sizes will increase with more training images.

* **Output files:**
* [seals\_training.tfrecord, seals\_validation.tfrecord, seals\_test.tfrecord] (~300MB\*, ~30MB\*, ~30MB\*, compressed images for training, validation and testing)
* seals\_model.pb ( ~ 300MB, compressed model file with architecture and weights)

\* approximate file sizes for ~ 5000 raw training images; file sizes will increase with more training images.

* + **Time for completion:** ~20 hours / GPU / combination of hyperparameter (running time will vary depending on batch size and number of epochs)
  + **Barrier between stage 1 and stage 2:** True
  + **Storage Space:** Training set (~500MB); TensforFlow model files (~300MB per version)
* **Stage 2** 
  + **Description**: Process input imagery and create a PostgresSQL database to manage input imagery and covariates. Input images need to be separated into multiple images with the correct dimensions for the CNN (299x299 pixels). To standardize resolution across images, cropped images will be supersampled or subsampled, depending on their off-nadir angles, to a 100x100m square. Store off nadir angles and latitude, longitude as raster to be used as covariates in later stages.
  + **Input:** WV3 orthorectified satellite images (‘.tiff’ files) within the Antarctic coastline.
  + **Output**: PostgresSQL database with georeferenced chopped down images.
  + **Barrier between stage 2 and stage 3:** True
  + **Storage Space:** ~200GB (images), ~3GB (database rasters)
* **Stage 3**
  + **Description**: Add auxiliary covariate data to PostgresSQL database from **Stage 2** to be used when dealing with imperfect detection in **Stage 8**. Each covariate is added as a raster, subsampled or supersampled to match the resolution of chopped down images. Covariates of interest include several sea-ice related features, sea surface temperature (SST), chlorophyll, distance to bathymetric features, bathymetry, time of the day and Julian day.
  + **Input:** Raster layers for environmental covariates.
  + **Output**: PostgresSQL database with environmental covariates.
  + **Barrier between stage 3 and stage 4:** False
  + **Storage Space:** ~3GB (database rasters)
* **Stage 4**
  + **Description**: Setup TensorFlow model to be deployed with multiple GPUs (40 Tesla K80s at seawulf). May or may not require setting up a local TensorFlow server, but ideally a SSH would suffice.
  + **Input:** Input images in postgresSQL database (**Stage 2**), TensorFlow model file for CNN trained on seal haul outs (**Stage 1**).
  + **Output**: Queue for processing input images using multiple GPUs.
  + **Barrier between stage 4 and stage 5:** True for continental scale. False at a local scale.
* **Stage 5**
  + **Description**: Classify input images as to whether or not they contain groups of seals (i.e. one or more seal).
  + **Input:** computation Queue.
  + **Output**: Raster with Boolean for presence of seals (attached to the postgres database from **Stage 2**.
  + **Barrier between stage 5 and stage 6:** True
* **Stage 6**
  + **Description**: Train a new CNN to count seals inside seal haul outs, similar to **Stage 1**, but with the caveat that this will work best with native need object detection (i.e. label many objects inside image) architectures (Inception Resnet is an object classification CNN, i.e. gives one label per image), such as YOLO9000 and Single Shot Multibox (SSD). Assembling a new training set for counting seals will require drawing bounding boxes around each seal in training images, which can either by semi-automated using a simple blob detector or outsource with services such as Amazon mechanical Turk.
  + **Input:** Training set with bounding boxes, Model architecture, input images with seals.
  + **Output**: Seal counts within images with seals (Raster uploaded to postgres database), TensorFlow model for counting seals.
  + **Barrier between stage 7:** True
* **Stage 7**
  + **Description**: Use R and JAGS to build Bayesian models for estimating total population size given potential detection failures from CNNs at **Stage1** and **Stage 6**.Model detection failures coming from three sources: 1) the seal haul out detection algorithm from **Stage 1** missed a seal haul out (false negative at the patch level) or flagged an empty patch as having seals (false positive at the patch level), 2) the seal counting CNN from **Stage 7** missed a seal (false negative at the seal level) or classified another object (such as a rock shadow) as a seal (false positive at the seal level) – detection failure – and 3) seals were in the water at the time of the image and therefore not available for detection – availability failure. Detection failure at the patch level will be modelled as a Bernoulli random process:

where:

Detection failure at the seal level will be modelled as a Binomial process with a Poisson term for false positives:

where:

Total population in a patch will be modeled using a negative binomial:

where:

* + **Input:** Covariates from **Stage 4**, Seal counts from **Stage 7**.
  + **Output**: Array with posteriors for total seal counts at patches.