Use Case Document: ASIFT Automated ground-control and ortho-rectification of non-/poorly-geolocated aerial photos

# Descriptions

* **Collaborator:** University of Colorado
* Dr. Mike Willis
* Dr. Mike MacFerrin
* John Ohman
* **Field**: Topography, Glaciology
* **Measure of success:**
* **Phase 1:** Successfully auto-generate keypoints (matching pixel locations) from non-geolocated (historical) airborne imagery when searching over a database of ortho-rectified WorldView Satellite imagery.
* **Phase 2:** From the keypoint matches and a high-resolution digital elevation model (DEM), generate a topographically-corrected, ortho-rectified image the original aerial photos.
* **Primary challenges:**
* ASIFT Algorithm has not yet been used for aerial & satellite imagery, primarily been used on small PNG imagery.
* Data volumes of WorldView Ortho-Imagery and DEMs are enormous, ASIFT search will require substantial computational demands
* **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 1a (one time only):** **CNN for seal presence/absence** a) developing customized training set for the CNN, b) estimate the correct hyperparameters, c) training the CNN
  + **Stage 1b (one time only):** **CNN for seal counting** 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:** Distributing computation to GPU(s)
  + **Stage 4 Haul out detection:** Classification of seal haul outs byTensorFlow
  + **Stage 5 Seal counting:** Seal counting
* **Stage 1a CNN for seal presence/absence**
  + **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)
  + **Storage Space:** Training set (~500MB); TensforFlow model files (~300MB per version)
* **Stage 1b CNN for seal counting**
  + **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.
* **Stage 2** 
  + **Description**: Input images need to be (1) standardized to a common resolution grid (supersampled or subsampled by bicubic transformation using OpenCV) and (2) divided into multiple smaller images (called ‘patches’) with the correct dimensions for the CNN (299x299 pixels). For our application, we want 299 x 299 pixels to be 100m x 100m, and we would want to subsampled to a 100m/299 = 0.334 cm grid, but the script should take in desired resolution as an input. Note that the resampling will depend on the off-nadir angle of the original image and so we will need to extract that metadata from the image file or input that as a parameter. (We will create a PostgresSQL database to manage input imagery, metadata (e.g., off-nadir angle, center coordinates) and covariates but this may reside outside the core pipeline.)
  + **Input:** WV3 orthorectified satellite images (‘.tiff’ files) within the Antarctic coastline.
  + **Output**: Georeferenced ‘patches’ and metadata. (We will use the metadata to create a PostgresSQL database that stores the metadata and the images themselves, though this may not be the way that other users want to manage the database and so we should keep the output of this Stage as generic as possible.) \*\*There may be EarthCube metadata standards that we need to abide here.
  + **Storage Space:** ~200GB (images), ~3GB (database rasters)
* **Stage 3 (Resource management issue, perhaps not considered its own Stage?)**
  + **Description**: Set up TensorFlow model to be deployed with multiple GPUs (40 Tesla K80s at Seawulf). (Local TensorFlow server useful?)
  + **Input:** (1) Input images from **Stage 2**, and (2) TensorFlow model file for CNN trained on seal haul outs (trained during **Stage 1**).
  + **Output**: Queue for processing input images using multiple GPUs.
* **Stage 4**
  + **Description**: Classify input images as to whether or not they contain groups of seals (i.e. one or more seal).
  + **Input:** Output from **Stage 3**. (i.e. input ‘patches’ and TensorFlow model)
  + **Output**: Table with two columns, one column for patch ID and additional columns for extracted features. In our case, our classification yields a simple 0/1 for the presence or absence of seals in that patch. (The PostgresSQL database provides the link that associates each Patch ID to the appropriate metadata (scene information, etc.).)

* **Stage 5**
  + **Description**: Detecting individual seals in patches that have been classified as containing seals in **Stage 4**. Seals are detected using the CNN developed in **Stage 1b**.
  + **Input:** Output from **Stage 4** and CNN from **Stage 1b**.
  + **Output**: Table with two columns: Latitude and Longitude of each detected seal.