Counting Multi-Class Steller Sea Lion Populations

Charles Lovering [*1] Ben McMorran [*1]

Abstract

Saving the declining Steller sea lions in the Western Aleutian Islands has been a focus of scientists at the NOAA Fisheries Alaska Fisheries Science Center. In order to understand factors that may be preventing the Steller's populations from resurging, NOAA collected aerial photographs and counted the different groups within the population. As part of a competition hosted by *Kaggle*, we developed Deep Learning techniques to automate the process. We experimented with both density counting and object detection approaches, but found little success.

1. Introduction

The population of Steller sea lions in the Aleutian Islands of Alaska has been declining since 1980s, and the cause is still unknown (Dahle & London, 2017). Although the decline has stabilized, the population has not recovered. To better understand these changes, scientists have launched several missions to capture these animals, collect vital information from them, and tag them with GPS receivers (Dahle & London, 2017; Dahle, 2017). Additionally, biologists manually count the population from thousands of high-definition aerial photographs (Figure 1). Counting is a time and labor intensive process, requiring four months to complete (Kaggle, 2017). In this paper we develop techniques to automate this process.

The task is complicated by the fact there are variable numbers and types of sea lions in each image (adult males, subadult males, adult females, juveniles, and pups). The images are also large, ranging from 15 to 20 megapixels, while the sea lions themselves are often no more than 50x50 pixels. We approached this problem as an object detection problem, and employed several techniques to simplify the learning process. This paper details the process in Section 3.

[*]Equal contribution [1]Worcester Polytechnic Institute, Worcester, MA. Correspondence to: Charles Lovering <cjlovering@wpi.edu>, Ben McMorran <bjmcmorran@wpi.edu>.

CS525 Special Topics Course Final Project, Worcester Polytechnic Institute. Copyright 2017 by the author(s).

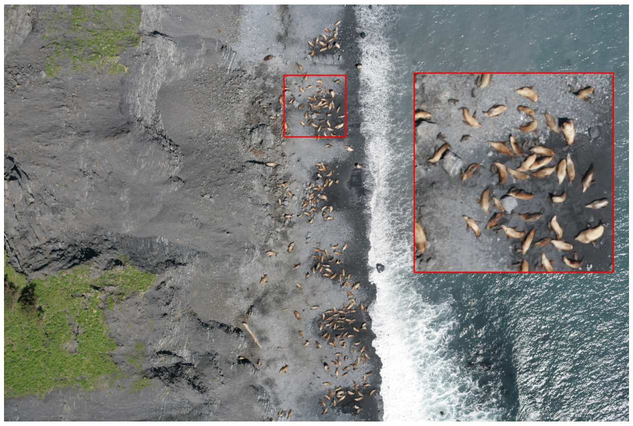

Figure 1. Example aerial image. Detail shows sea lions on beach.

1.1. Research contributions

In this paper we explored the challenges related to applying object counting techniques to real-world high resolution imagery, finding that scalability becomes a concern at large sizes. Both training and prediction runtime increase dramatically for well-known techniques. Additionally, techniques that work well for small image patches can struggle to achieve the same performance on full-size images.

2. Related Work

Several related techniques helped inform our approach.

2.1. Density Counting

(Walach & Wolf, 2016) developed a technique for counting objects in images by creating a density map of a training image, training a CNN to generate density map patches given training image patches, and then sliding the CNN across a test image to generate a predicted density map. Finally, the predicted density map is summed to obtain the predicted object count.

While this approach is robust to partial occlusions and provides some degree of object localization, it requires that the density kernels for each object be known in advance. This assumption is appropriate for crowd counting where the down direction can be safely assumed, but unfortunately aerial photographs can be taken in any orientation and sea lions do not have rotational symmetry from above, causing frequent errors in the density map.

2.2. Survey of Modern Object Detection

Although this work does not require object detection, the task of locating and classifying variable number of objects in a an image, it is closely aligned. An approach to this problem is to apply a detection model and then count the resulting objects. We review recent techniques for object detection to both determine which of these techniques to first try and to inform any derivative models we build.

2.2.1. RCNN

RCNNs use a multistage process (Girshick et al., 2013). Via an exterior Region of Interest proposal system, it first trains a CNN on the proposed object proposals. It then uses several binary SVMs on these features to classify proposals. Then, a regression head operates on each of these features to predict the precise bounding box. The training is expensive. Further, the object detection in test time is also slow as it requires an external system to generate the proposals. The CNN it uses is VGG (Simonyan & Zisserman, 2014).

2.2.2. Fast RCNN

The Fast RCNN improves upon the RCNN by streamlining the process of training the classification and the bounding box regression. This allows for a faster train and test time. One of the key innovations to this process was the using a Region of Interest (RoI) Pooling Layer that max pools any region of interest projected on the feature map into a fixed size feature map (Girshick, 2015). Each of these is then fed into classification and regression heads that consist of several fully connected layers.

2.2.3. Faster RCNN

The Faster RCNN again follows the pattern of increased simplicity and streamlines the process by using a Region Proposal Network (RPN) that internally proposes Regions of Interest rather than using an external source (Ren et al., 2015). This allows the entire process to be trained in a single stage. RPN is implemented by sliding a window across the feature map and extracting a vector. For every sliding window location there are k proposals, and for each of these the RPN regress two scores for softmax classification as a positive or a negative example, and 4 coordinates that describe the region (x and y of top left and width and height). Each of these k 'anchors' are associated with a specific scale and aspect ratio. The original work used three different scales and ratios a ($k = 9$).

2.2.4. Mask RCNN

The Mask RCNN (He et al., 2017) is the next generation of the Faster RCNN framework that is also capable of Object Instance Segmentation. It does so by adding a third head that predicts a mask for each region of interest (in parallel to the classification and regression heads. There are some implementation details necessary to update the Region Proposal Network, but beyond the scope of this survey. Mask RCNN's success with detection is a by product of its performance in segmentation.

3. Proposed Method

We approached the problem in two different ways. Object detection methods (Section 2.2) suffered from a lack of accurate ground truth bounding boxes, because Kaggle only supplied location information for each sea lion. Using the information provided we pursued *divide and conquer* methods to break down the problem into smaller tasks.

3.1. Binary Thumbnail Classification

In our first approach, we trained a CNN to classify image patches as containing or not containing a sea lion. We used 128x128 patches centered on each sea lion as positive examples, and randomly selected patches at least 100 pixels away from any sea lion as negative examples. Figure 2 illustrates the architecture. All activation functions use the ReLU. We used many small 3x3 convolutions, emulating the approach used by the successful VGG16 image classification network (Simonyan & Zisserman, 2014).

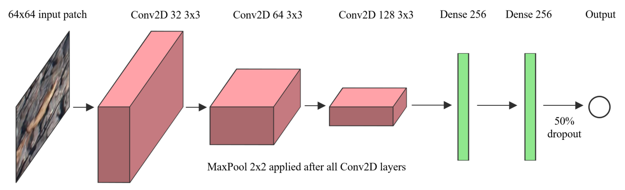

Figure 2. Architecture of the binary classification network. Input patches were originally 128x128, but are resized before being fed to the network.

While ideally the network would also be able to classify the type of sea lion, due to time and computational constraints we focused on this simpler binary sub-problem. Pups were not used as positive examples due the difficulty in visually distinguishing them from rocks. They are typically darkly colored and tend to hide under females.

Once trained, this CNN can be iteratively applied to all patches in a full-size input image, producing a feature map similar to the density maps used by (Walach & Wolf, 2016). This map can either be summed to directly estimate the population count, or another CNN can be trained on the density maps to produce a final estimate. We experimented with both approaches.

3.2. Sliding Window

This approach consisted of partitioning images and attempting to use a network to predict the population in each of the partitioned images.

3.2.1. Partitioning the Images

We split each image into a grid equally sized partitions (624 x 416) and assigned rough bounds based on location data. An example of a sliced image with the approximate bounding boxes around each sea lion is shown in Figure 3.

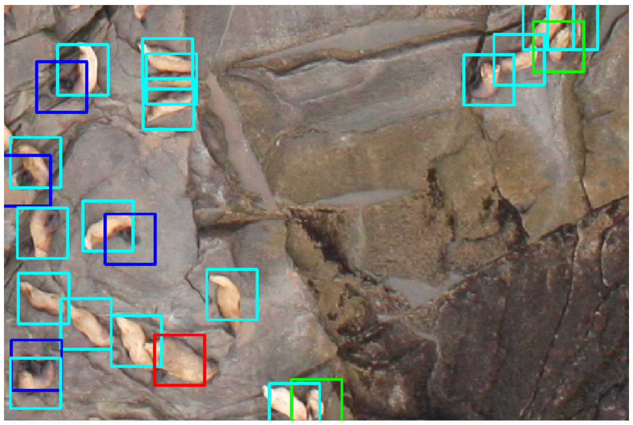

Figure 3. Ground-truth bounding boxes on a sliced image. Colors correspond to class: pup, male, subadult male, female.

3.2.2. Utilizing Convolutional Neural Networks

The first approach was to train a CNN to regress the 5 population numbers for each image and combine predictions for the larger image. Although naive (we shall refer to it as SimpleNet), it is inspired by the example of the powerful results in (Girshick, 2015; Ren et al., 2015; He et al., 2017). Each of these networks is ultimately *powered* by a single deep convolutional network that creates a rich feature map. We hypothesize that a properly trained CNN will be able to generate a rich enough feature map to have some results in predicting the count of the population. The methods described in Section 2.2 also allow it to predict locations, but it is unclear this is necessary for counting. For these models we used ResNet (He et al., 2015) as our core CNN, but the choice between it and other networks (VGG) have not been tested, and other choices may perform better.

Preliminary investigation found that the class imbalance in split pictures was severe; most partitioned images had no sea lions. Simply weighing the classes differently did not seem to be getting the requisite results. Thus, in parallel we developed a MultiLoss Regression Network described immediately below in Section 3.2.3.

3.2.3. MultiLoss Regression Network

The pipeline for this framework is shown in Figure 4. We shall refer to this architecture as MultiNet.

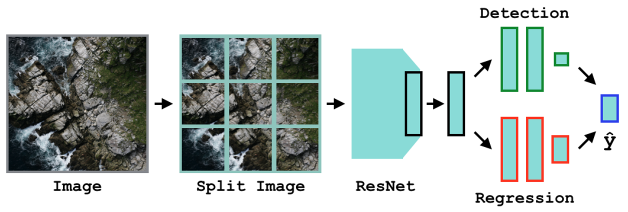

Figure 4. MultiNet. Each slice of the image is processed by the network separately. The detection head determines whether or not to use the results from the regression head. Each slice's results are aggregated offline.

This method is a simple addition to the framework detailed in 3.2.2 but attempts to handle the issue of unbalanced classes. Similar to (Ren et al., 2015) which uses the same CNN for multiple tasks, we utilize a loss function that trains the network to detect whether or not there are any sea lions in an image, and also predict the population. This allows the network to train in a sensible manner. This is because around 90% of split images have zero instances of a sea lion, and MSE leads the network to underestimate the true number of sea lions. At first we attempted to change the training image distribution. However, this alone is a volatile method. Instead, we developed two streams of the network, one that detects seals and one that predicts the population. Here *i, j* map to a grid position of the image indexed by *n*. The prediction for a single image aggregates these results from the network:

$$\hat{y}_n = \sum_{i,j \in W_n, H_n} I(d_{i,j}) * (r_{i,j}) \tag{1}$$

For each image slice the model uses the same feature map to determine whether or not there are any sea lion present, and then predicts the sea lion population. These loss is then a combination of a softmax for the detection and MSE for the regression. The network's final result also has its own MSE loss. Some of the intuition behind this: if the network is able to fully fit itself to accurately predict the correct number of sea lions given the assumption that sea lions are present, then perhaps allowing it to do so and additionally checking if the assumption is true may provide better results than the alternative.

Training this network is left to Section 4.2.2. Given time, we would experiment with optimizing final loss, separating the two losses (regression and detection), or simply including all three losses. We first have only tested the latter approach and more validation is needed.

4. Experiment

We built our models using Keras (Chollet et al., 2015).

4.1. Binary Thumbnail Classification

Using the training data set provided by Kaggle, we extracted 113,511 128x128 patches evenly split between positive and negative examples. 20% of these examples were reserved for a validation set. An explicit test set was not required because Kaggle already maintains hidden test set data for final verification.

We trained model described in Section 3.1 using Adam optimization (Kingma & Ba, 2014) and cross-entropy loss for 50 epochs at three different learning rates: 10^{-2}, 10^{-3}, and 10^{-4}, finding that a learning rate of 10^{-4} was best. We also experimented with a wider network, but found that its performance was no better. After training, the best model had a validation accuracy of 98.65% with an AUC of 0.9980.

We next took sequential patches from a full-size image with a stride of 64 pixels to produce a feature map. We predicted the total number of sea lions in each class using both a simple linear regression model based on the total area covered, and a more complex CNN regression model based directly on the feature map. The CNN model used 8 3x3 convolutions and 4 3x3 convolutions in the first and second layers, plus a single 32 unit hidden layer. Feature maps were resized to 32x32 before being processed.

4.2. Sliding Window

4.2.1. SIMPLENET

Each of these networks take a large amount of time to train due to sheer amount of data. A single epoch would take approximately a day. This model was instead trained only on images with sea lions (10,000 images) in them which takes about 4 hours per epoch for 10 epochs. It was then tuned on 10,000 images negative and positive. During test time it processes each partitioned image separately and sums the results. The model was initialized with the weights from ImageNet.

4.2.2. MULTINET

We started the training by transferring the weights of SimpleNet. From here, the model was trained on 30,000 images sampled without bias from positive and negative examples (both containing and not containing sea lions).

5. Results

In accordance with Kaggle rules, performance is measured using root mean squared error (RMSE) averaged across all images. As a baseline, guessing all zeros achieves an RMSE of 29.08704, while guessing the mean for each sea lion type achieves an RMSE of 27.51308.

5.1. Binary Thumbnail Classification

Neither of the approaches outlined in Section 4.1 achieved an RMSE better than guessing the mean. The linear regression model had an RMSE of 30.32413, while the CNN based model was even worse, with an RMSE near 50. The poor CNN performance is likely due to overfitting on the relatively small (948) training set. Adjusting the classification threshold to get a true positive rate of 95% and a false positive rate of 0.30% did not significantly affect the results.

5.2. Sliding Window

Currently the results we have for this do not hold a lot of generalized meaning; with the MultiNet we achieved an RMSE of 78.23 on a validation set and with the SimpleNet resulted in an RMSE of 444.023. These results show that little was learned from the data. An explanation for this is that it may be due to the fact that the model saw relatively little of the entire dataset of sliced images. In relatively little training the loss of the MultiNet severely dropped from the SimpleNet. Further experimentation and tuning is necessary to fully rule out this line of model. This includes: increase the numbers of epochs on the entire dataset, cross validate different core CNNs (VGG and others), and cross validate using different losses. Very late in the process an issue was found in the detection head. The way that the softmax interacted with one output node was to always predict positive. This weakens the network as if all the labels were wrong. This has been fixed, but the results above come from the weakened network.

6. Discussion

To better understand the poor performance of the binary classifier on the full data set, we plotted the top 10 classification errors for positive and negative examples (Figure 5).

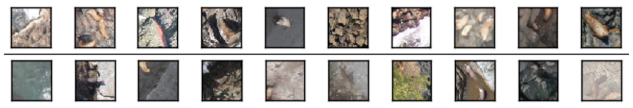

Figure 5. Top 10 classification errors. Top row has is not sea lion ground truth; bottom row is sea lion ground truth.

It appears that the network still struggles to find sea lions in dark or occluded areas, as well as underwater. Several sea lions actually appear in the negative ground truth examples. These are likely sea lions that missed human labeling. Rough rocks are also labeled as sea lions, which is particularly problematic for large swaths of land of this texture. Using a classification approach to counting is sensitive to the large class imbalance present in patches taken from a full-size image. Even small errors can be quickly amplified and dwarf any useful signal.

While utilizing deep learning models in real-world contexts we encountered a host of issues. The corpus of data we dealt with was over 100GB. Managing this with limited resources was a hindrance.

6.1. Tools and Techniques

We ran binary classification experiments on a Microsoft SurfaceBook, and all other experiments on the Google Cloud Compute engine. It was easy to use with no issues and is an example of the democratization of powerful hardware. We were able to use a free account with 2 GPUs. Working with Keras was also extremely refreshing. It is a slight step above TensorFlow in terms of abstraction, and provides a wide range of easy to use features. Most importantly its documentation outclassed (from what we have seen and used personally) other libraries in terms of extensiveness, examples, and clarity.

7. Conclusions and Future Work

While none of our approaches was successful on the regression task, each provided experience for how to deal with large, complicated, varying, and potentially noisy data. The models described in Section 3.2 show some potential but require further validation. Another area for future experimentation is directly applying object detection frameworks to this problem.

References

Chollet, François et al. Keras. https://github.com/fchollet/keras, 2015.

Dahle, Shawe and London, Josh. Dispatches from the field: Studying at-risk harbor seals in western aleutians, 2017. URL https://www.afsc.noaa.gov/Science_blog/Hs_2016_main.htm.

Dahle, Shawn. Harbor seal research in the aleutian islands, 2015, 2017. URL https://www.afsc.noaa.gov/Quarterly/JAS2015/divrptsNMML3.htm.

Girshick, Ross B. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL http://arxiv.org/abs/1504.08083.

Girshick, Ross B., Donahue, Jeff, Darrell, Trevor, and Malik, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013. URL http://arxiv.org/abs/1311.2524.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask R-CNN. *ArXiv e-prints*, March 2017.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Kaggle. Noaa fisheries steller sea lion population count, Mar 2017. URL https://www.kaggle.com/c/noaa-fisheries-steller-sea-lion-population-count.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ren, Shaoqing, He, Kaiming, Girshick, Ross B., and Sun, Jian. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL http://arxiv.org/abs/1506.01497.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Walach, Elad and Wolf, Lior. Learning to count with cnn boosting. In *European Conference on Computer Vision*, pp. 660–676. Springer, 2016.