Identifying Elephant Seals in Drone Imagery

Julian Davis - Cal Poly San Luis Obispo
jdavi104@calpoly.edu

June 5, 2020

1 Introduction

Marine biologists are tasked with manually counting and identifying elephant seals in the San Simeon area. They can then use this information to determine trends in seal populations, migration, and reproductive success. But this process is tedious and time consuming. The goal of this senior project is to automate this process. We want to replace the marine biologists with drones that fly over the elephant seals periodically to capture images and video. Then, we will run a machine learning pipeline which given the drone data, gives us accurate estimates of the seal population.

2 Data

We don't currently have drone data of elephant seals over the San Simeon area, but we found existing data from AnoNuevoResearch. This drone data is similar to the data that will eventually be collected, so we determined it was a good starting point. The drone data by itself, however, is not enough for supervised machine learning. We also need data giving us information about where the seals are in each image. To do this, we found a tagging tool that allows us to create bounding boxes around each seal. The corresponding bounding box data can then be exported as a XML file. With this tagging method, my partner and I each tagged ten images of seals each to get twenty tagged images of seals. But twenty images is not sufficient for our training purposes. We needed a way to get more tagged images efficiently. To do this, we taught Dr. Liwanag's class of marine biologist students how to use the tagging tool and had each of them tag five images. After compiling all the data, we had eighty images and a total of 5848 elephant seals tagged, giving us enough data to properly train a model.

Figure 1: Drone Image of Elephant Seals

3 Approach

We've established that we want to use supervised machine learning to get estimates of the seal population. We started by breaking down the problem of estimating the seal population into smaller parts. To get estimates of the seal population, we first need to be able to estimate the number of seals in a drone image. To estimate the number of seals in a drone image, we first need to be able to identify the presence of a seal. Identifying the presence of a seal is a good starting point because it breaks the problem down to a recognition problem.

Image recognition is a tangible problem to solve because it is precisely what convolutional neural net-

works (CNNs) are designed to do. CNNs are a category of Neural Networks that have proven effective in classifying images. CNNs expect and preserve the spatial relationship between pixels by learning internal feature representations using small squares of input data. Features are learned and used across the whole image, allowing for the objects in the images to be shifted or translated in the scene and still detectable by the network. All in all, CNNs are very powerful and are the perfect mechanism for what we're trying to accomplish.

We know we want to use CNNs for image recongition, but we can't just feed a model a drone image (like the one in Figure 1) and it's corresponding bounding box data. We need a way to divide the drone images into subimages. The idea behind this is to divide each image such that every seal within it has a chance of being completely within a subimage. Then, we create a network that can recognize the presence of a seal. After achieving reasonable recogition, we want to transition to a model that can classify the subimages into one of three categories: there are no seals in the subimage, there is a partial seal in the subimage, and there is a complete seal in the subimage. Using this network, we can simply count how many times it predicts a complete seal (in the subimages) to obtain an estimate for the number of seals (in the image).

4 Splitting and Relabelling

First, we perform the "splitting". We divide each image into subimages using split_image() and parameterize this function so that it is capable of shifting. Shifting is what allows us to fine-tune the splitting such that every seal within an image has a chance of being completely within a subimage. After testing different splits and shifts, we determined the split and shift that was most optimal: splitting the image into 150 by 150 pixel subimages and shifting over 75 pixels each run through. Figure 2 demonstrates what the first "split" would look like and Figure 3 demonstrates what the second "split" would look like. These figures demonstrate how we are generating the subimages, although many more "splits" were performed.

Figure 2: Split #1 on Sample Drone Image

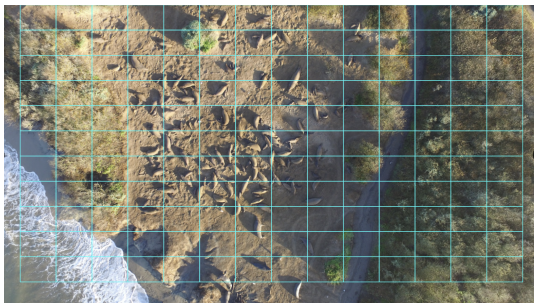

Figure 3: Split #2 on Sample Drone Image, Shifted by 75 pixels from Split #1

We have the images needed to train the CNN, but we still need the labels.

Next, we perform the "relabelling". We must ensure that each subimage has the appropriate tagging data derived from the original tagging data. To do this, we take the following steps:

1. We determine which bounding boxes are found within a sub image
2. We determine the new coordinates for the bounding boxes
3. We determine what percentage of the seal is found within the subimage, calculated by the percentage of the bounding box present in the subimage. This will be used later on to parameretize the network.

Table 1 demonstrates what the bounding box data looks like for a seal in an image while Table 2 demonstrates what it looks like in a subimage.

Table 1: Image Bounding Box Data for Seal

file_num	0001
label	adult male
xmin	2240
ymin	1310
xmax	2319
ymax	1379

Table 2: Subimage Bounding Box Data for Seal

file_num	0001
label	adult male
xmin	150
ymin	140
xmax	150
ymax	110
percent_seal	0.073381

The classification remains the same, the coordinates are updated, and the seal percentage is calculated. Finally, we save the subimages and corresponding tagging data in **images.npy** and **bb_data.npy**, respectively. Each item in **images.npy** is a subimage. Each item in **bb_data.npy** is either none if it's corresponding subimage doesn't contain a seal or is a data frame containing the relevant meta data (as exemplified in Table 2).

5 Training the Network

Our primary goal is to train a convolutional neural network to perform tertiary classification where it can classify an image as belonging to one of three categories: "No Seal", "Partial Seal", or "Full Seal". But, this is not an easy task. It is difficult for the network to distinguish between "No Seal" and "Partial Seal" and between "Partial Seal" and "Full Seal". In order to break this problem down further, we began our training process with the simplest case: seal recognition. Given an image, can we determine if it contains a seal or not (classifying between "Seal" and "No Seal"). We first sought to achieve high accuracies with recognition. We then sought to understand the thresholds at which partial seals are classified as "No Seal" and the thresholds at which partial seals are classified as "Seal". Knowing these thresholds will give us insight into how the network is predicting and help us transition to tertiary classification later on.

5.1 Preprocessing

Preprocessing is a common practice used for altering the input space before it enters the network. We took several key preprocessing steps in an attempt to optimize the network's performance.

First, we converted all RGB values of the images from the 0 to 255 range into the range between 0 and 1 by dividing the whole tensor by 255. Converting the values to a 0-1 scale helps performance in neural networks. Empirical evidence suggests that, "neural networks tend to work best with data in the range of either 0 to 1 or -1 to 1" (NNFS).

Second, we filtered the images to get a dataset where 40% of the images contained a seal. This means we intentionally threw out a significant number of images that don't contain seals. Doing this reduces both the noise in the dataset and the computational time required to train the network.

Third, we filtered out images in the dataset containing a seal percentage greater than 0 and a seal percentage less than *threshold_min*. Let's call these filtered out images the "Partial Seal" images. By removing the "Partial Seal" images, we give the network a much better chance of recognizing seals with high accuracy. This is a good first step towards understanding the network and ensuring that, at the very least, it can recognize seals.

5.2 Grid Search

We developed a comprehensive grid search that will allow us to explore many different network architectures and hyper-parameters. Our grid search is performed over the following variables:

- *threshold_min*: the minimum percentage to indicate the presence of a seal
- *threshold_max*: the maximum percentage to indicate the presence of a partial seal; used for

tertiary classification only

- *cnn_blocks*: number of conv-conv-pool (CCP) blocks
- *dense_layers*: number of dense layers
- kernel_size: specifies height and width of convolution window
- *strides*: specifies the strides of the convolution along the height and width
- *dropout_flag*: a boolean flag indicating whether to add a dropout layer

5.3 Metrics

Choosing what metrics to include is extremely important in understanding our model's performance and behavior. In order to calculate various metrics of interest, we have two different held-out test sets of data. The first test set is a standard test set, which we obtain from a 90-10 train test split. We use this test set to calculate the following metrics: accuracy, precision, recall, f1 score, and a confusion matrix.

The second test set is not standard. We obtain the second test set by collecting another train-test split, which we call the "no filter" split. The "no filter" split is a train-test split that does not filter out the images with a *seal_percentage* greater than 0 and less than *threshold_min*. We use the test set from the "no filter" split to create a bin plot of the percentage of images predicted as a "Seal" vs *seal_percentage*. These bin plots are demonstrated in Figures 4 and 5. Creating this separate test set allows us to see how the network is performing on images from all ranges of *seal_percentage*, not just those filtered out by *threshold_min*. We plan on using these visuals, specifically, to identify potential thresholds for tertiary classification.

6 Results

6.1 Seal Recognition with a *threshold_min* of 70%

Tables 3, 4, and 5 correspond to our best model for binary classification with *threshold_min* at 70%. Setting *threshold_min* to 70% means that we are only training on images with no seal and images that have *seal_percentage* greater than or equal to 70%. By throwing out images containing "Partial Seals", we drastically improve the network's ability to recognize seals. Table 4 shows promising recognition metrics. Most notably we achieve a test accuracy greater than 90%. Table 5 shows a promising confusion matrix. We don't see much discrepancy between the false negatives and the false positives, indicating that the network's predictions are balanced.

Table 3: Model Architecture

CCP Block(s)	2
Dense Later(s)	2
Filter Mutliplier	0x
Kernel Size	3
Strides	2
Threshold Min	.70
Threshold Max	N/A
Dropout?	Y

Table 4: Test Metrics

Accuracy	.9004
Precision	.9005
Recall	.9004
F1 Score	.9005

Table 5: Confusion Matrix

T/P	Positive	Negative
Positive	1080	95
Negative	101	692

Figure 4, as described in Section 5.3, is created using a separate test set that does not filter out the images with a *seal_percentage* greater than 0 and less than *threshold_min*. The steady increase from one bucket to the next in the plot matches our intuition. As more of a seal is found within the subimage, the chances of it being detected increases. The plot also indicates what thresholds could be promising for tertiary classification. We see a significant jump from

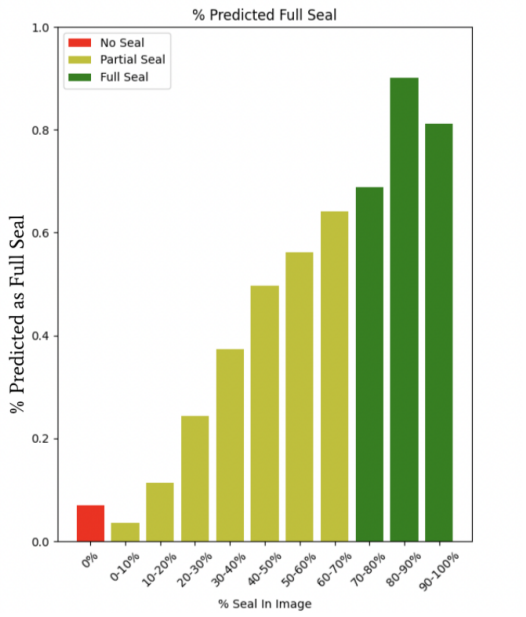

Figure 4: Bin Plot of Percentage Predicted As a Full Seal vs Percentage of the Seal within Image Using Threshold Min at 70%

the 10%-20% to the 20%-30% bucket and from the 20%-30% bucket to the 30-40% bucket. This indicates that a *threshold_min* between 20 and 30 would likely be optimal for tertiary classification, drawing a boundary line between what images should be classified as "No Seal" and what should be classified as "Partial Seal".

6.2 Seal Recognition with a *threshold_min* of 95%

Tables 6, 7, and 8 correspond to our best model for binary classification with *threshold_min* at 95%. Table 7 shows promising recognition metrics, although the results are not as impressive as those achieved with *threshold_min* at 70%. This is a little surprising. Our intuition tells us that increasing *threshold_min* would increase the network's ability to recognize seals: having more of the seal in the image should make it easier for the network to recognize it's presence. A potential culprit for this discrepancy is the volume of training data available. Specifically, there are 343 more images with seals using a *threshold_min* of 70% as compared to 95%. If the data volumes were equal, we suspect that using higher *threshold_min*'s would outperform lower *threshold_min*'s.

Table 6: Model Architecture

CCP Block(s)	3
Dense Later(s)	5
Filter Mutliplier	0x
Kernel Size	3
Strides	2
Threshold Min	.95
Threshold Max	N/A
Dropout?	Y

Table 7: Test Metrics

Accuracy	.8713
Precision	.8710
Recall	.8712
F1 Score	.8705

Table 8: Confusion Matrix

T/P	Positive	Negative
Positive	729	103
Negative	68	429

Figure 5, similar to Figure 4, demonstrates a steady increase in percentage predicted as a seal as *seal_percentage* increases. We see similar jumps from the 10%-20% to the 20%-30% bucket and from the 20%-30% bucket to the 30-40% bucket. We also see a significant jump from the 80%-90% to the 90%-100% bucket. This indicates that a *threshold_max* around 90 may be optimal for tertiary classification, drawing a boundary line between what images should be classified as "Partial Seal" and what should be classified as "Full Seal". Although it should be noted that the results of this plot are less impressive than in the previous plot. With a more extensive grid search and more data, we would hope to find similar results with a more accurate model. This would serve as convincing evidence that a *threshold_max* around 90 is the right choice for tertiary classification.

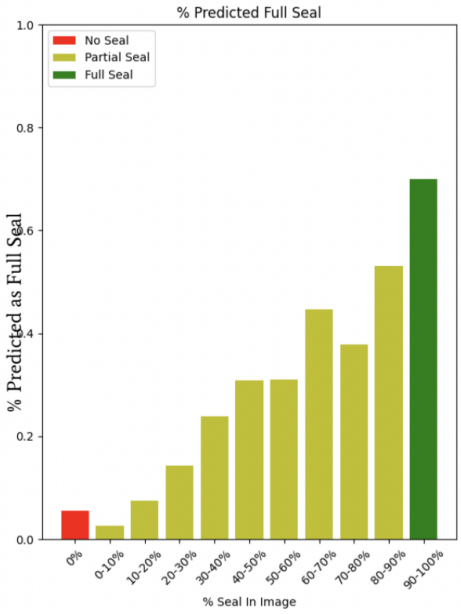

Figure 5: Bin Plot of Percentage Predicted As a Full Seal vs Percentage of the Seal within Image Using Threshold Min at 95%

7 Conclusion

The goal we initially set for this project was to develop a machine learning pipeline to give us estimates for the seal population. We did not reach this goal. But the results we've achieved so far show promise. We achieved accuracies greater than 90% on the task of seal recognition using a *threshold_min* of 70%. Thus, proving that it is possible for a network to recognize seals at a high clip. Elephant seals do a good job camouflaging with the habitat, so just being able to recognize them is a feat on it's own.

The next steps for this project is to transition from the task of recognition to tertiary classification. Can we classify an image into one of "No Seal", "Partial Seal", or "Full Seal"? We've already started the process of identifying thresholds, as discussed in our analysis of Figures 4 and . The goal is to find a reasonable *threshold_min* and *threshold_max* that can be used to classify the images into one of the three categories. A lot of the difficulty lies in the boundary cases. For instance, can we really expect the network to distinguish an image with a *seal_percentage* of .69 as "Partial Seal" from an image with a seal_percentage of .70 as "Full Seal"? The practical answer is no. But the overall objective is to use tertiary classification to obtain **estimates** of the seal population. Here is a quick recap for how we plan to extract population estimates from tertiarty classification:

1. Divide each image such that every seal within it has a chance of being completely within a subimage.

2. Create a network that can perform tertiary classfication reasonably well.

3. Using the network, count how many times the networks predicts "Full Seal" in the subimages to obtain an estimate for the number of seals in the original drone image.

Using this methodology, the end game is automate the process of estimating seal populations.

8 References

Y. Le Cun, B. Boser, Et al. "Handwritten Digit Recognition with a Back-Propagation Network".

Kunihiko Fukushima. "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position".

Jiuxiang Gua, Zhenhua Wang, Et al. "Recent advances in convolutional neural networks".

Harrison Kinsley, Daniel Kukieła. "Neural Networks from Scratch (NNFS)". https://nnfs.io

Dan Kong, Doug Gray, Hai Tao. "A Viewpoint Invariant Approach for Crowd Counting".

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks".

Victor Lempitsky, Andrew Zisserman. "Learning To Count Objects in Images".

Thomas Moranduzzo, Farid Melgani. “Automatic Car Counting Method for Unmanned Aerial Vehicle Images”.

9 Appendix

9.1 Results for Seal Recognition with a *threshold_min* of 30%

Table 9: Model Architecture

CCP Block(s)	3
Dense Later(s)	2
Filter Mutliplier	0x
Kernel Size	3
Strides	2
Threshold Min	.30
Threshold Max	N/A
Dropout?	Y

Table 10: Test Metrics

Accuracy	.8467
Precision	.8489
Recall	.8467
F1 Score	.8439

Table 11: Confusion Matrix

T/P	Positive	Negative
Positive	1745	343
Negative	138	912

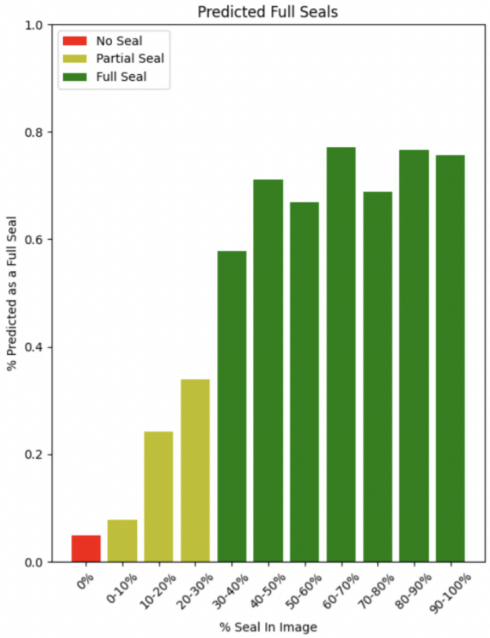

Figure 6: Bin Plot of Percentage Predicted As a Full Seal vs Percentage of the Seal within Image Using Threshold Min at 30%