Machine Learning Engineer Nanodegree
Capstone Project

Adilbek Madaminov
February 12, 2019

Definition

Project Overview

An important part of whale conservation efforts is to monitor and log whale movements. Scientists use photo surveillance to monitor whale activity, and for the past 40 years, most of the work identifying whales in photos has been done manually by individual scientists. This has left a huge trove of data untapped and underutilized.

“Humpback Whale Identification” [1] is a live Kaggle competition to identify individual whales from the images of their tails, with nearly 1,800 participants at the time of writing. Given a set of training images and whale Ids that they represent, this competition challenges to build an algorithm that best identifies whales in new images. (Kaggle is an online community of data scientists and machine learning experts, owned by Google, that is most known for real-world data science competitions)

Interestingly, this is the second iteration of this competition, after its first version completed 8 months ago (the current iteration has more labels to identify). This presents an opportunity to review some of the methods and results used in the first iteration. Most notably, the winning kernel is made public and describes the strategy behind his winning score [2]. Reviewing this kernel shows that the strategy involved a siamese neural network, an advanced machine learning algorithm that I will describe and ultimately attempt to apply in this project. It also shows that this was a challenging problem because the top score was 0.78 (out of 1.0) and it dropped very fast with the next best score equal to 0.65.

In my further review of past work and academic literature, I found that this competition is most likely an example of one-shot learning, a category of machine learning problems with one or very few examples per label available for training. The following is a select list of works on this subject that I found useful:

- "Siamese Neural Networks for One-shot Image Recognition”, Gregory Koch et al., 2015 [https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf]
- “Matching Networks for One Shot Learning”, Oriol Vinyals et al., 2017 [https://arxiv.org/pdf/1606.04080.pdf]
- “One-shot Learning with Memory-Augmented Neural Networks”, Adam Santoro, 2016 [https://arxiv.org/pdf/1605.06065.pdf]
- "One shot learning explained using FaceNet”, Dhanoop Karunakaran, 2018 [https://medium.com/intro-to-artificial-intelligence/one-shot-learning-explained-using-facenet-dff5ad52bd38]

Problem Statement

This is a multi-class classification problem to build an algorithm that best identifies individual whales in the test set of nearly 8,000 images. The performance of an algorithm is evaluated on the submission file that maps each of the test images to a whale identification code.

The algorithm is expected to use the training set of about 25,000 images and labels to learn identities of about 5,000 different whales. CNNs, or convolutional neural networks, would typically be the best tool for solving problems of this nature. However, this competition presents a particular challenge for using a standard approach – the scarcity of training images (most labels have only one or two training images). Because CNNs normally need hundreds or even thousands of images per label to train, a successful solution in this competition will almost certainly require a more advanced approach.

In this project, I attempt to apply one such advanced approach – a siamese neural network [3]. Siamese networks were introduced in the early 1990s but gained most of their popularity and improvements only recently, possibly after the 2015 paper by Gregory Koch et al. [4]. Siamese networks are designed for "one-shot" image recognition problems, i.e. problems with very few examples per label.

Metrics

The most natural choice of metrics in this project is the competition's official score (MAP@5) as detailed at the following competition link:

https://www.kaggle.com/c/humpback-whale-identification/data

The competition allows to make up to 5 predictions per image; this metric is an average of scores across all images, where an image scores 1.0 if the correct label is predicted in the 1st of the five predictions, 1/2 if in the 2nd, 1/3 if in the 3rd, 1/4 if in the 4th, 1/5 if in the 5th, and 0 if none of the five predictions are correct [5].

This metric is an optimal choice mainly because it is Kaggle's official measure of a model's performance, used to evaluate all submissions. Using this metric makes it possible to compare my results to those of all other participants, in addition to measuring my own progress over time.

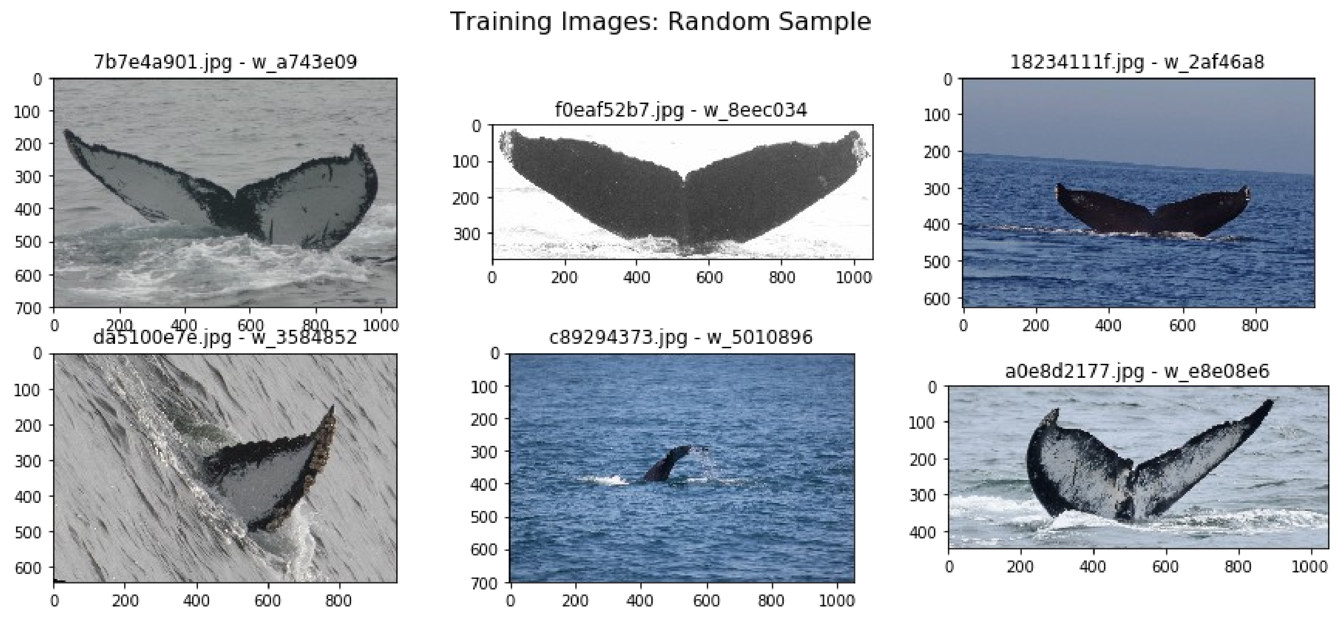
Additionally, I make use standard of metrics available in keras such as accuracy and crossentropy loss. These are great in intermediate stages of the training process, while the official competition score can be used as the final measure of a model's performance.

Analysis

Data Exploration and Visualization

(See notebook "data_exploration.ipynb" for code used to obtain information for this section)

The competition provides two folders of images: one with 25,361 training images and one with 7,960 testing images [6]. Additionally, the competition provides file "train.csv" that maps each training image to a corresponding whale Id. The following is a random sample of the training images, showing the file name and whale Id in the title:


Training Images: Random Sample
7b7e4a901.jpg - w_a743e09
f0eaf52b7.jpg - w_8eec034
18234111f.jpg - w_2af46a8
da5100e7e.jpg - w_3584852
c89294373.jpg - w_5010896
a0e8d2177.jpg - w_e8e08e6

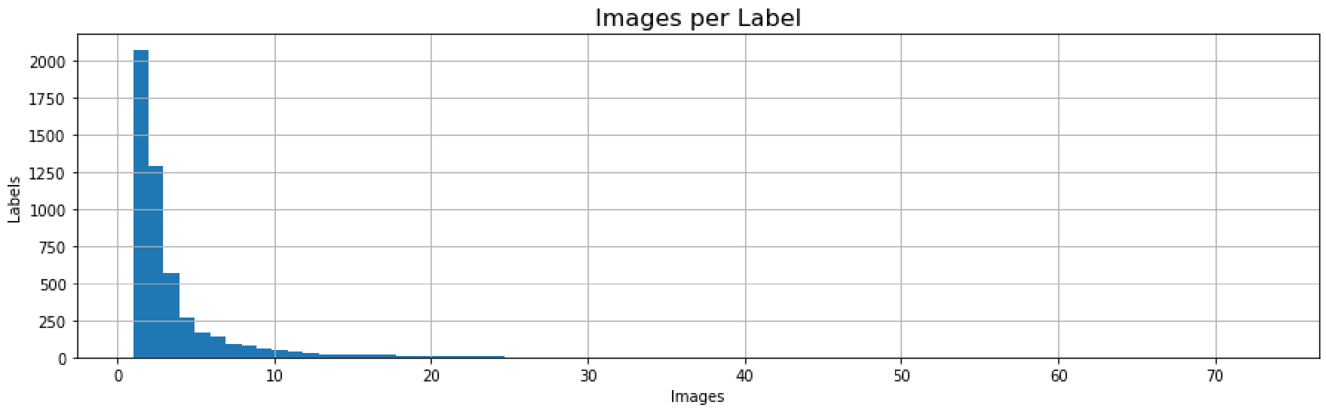
As can be noted even from this small sample, images vary in size, shape, and color mode, and some of the images are identified with a special label “new_whale”, indicating that they are not really identified. A more detailed analysis of the properties of the image files shows the following:

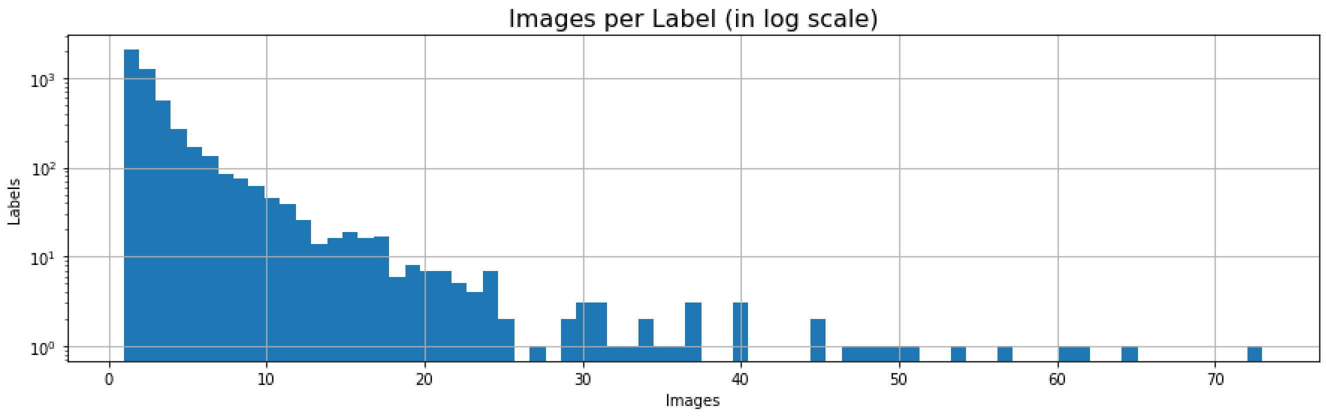
```
Training images:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Width:   mean: 992.3,   std: 267.2,   min: 77,    max: 5959
Height:  mean: 512.9,   std: 156.1,   min: 30,    max: 1575
RGB:     21975 (86.6%)  Grayscale: 3386  (13.4%)

Testing images:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Width:   mean: 997.2,   std: 280.0,   min: 126,   max: 5959
Height:  mean: 521.6,   std: 159.7,   min: 66,    max: 1199
RGB:     7137 (89.7%)   Grayscale: 823  (10.3%)
```

On average, the images appear to be mostly in color and of shape 1000 pixels wide by 500 pixels tall. But there is also a clear variation in both the color mode and size: some images are in grayscale, and some have odd shapes with very small or very large height and width.

A significant number of images is labeled as “new_whale” or effectively unlabeled. This is unusual in two ways: 1) these images are given as part of the training set where normally *every* image is expected to represent some label, and 2) these images make up a significant portion of the training set: 38% (9,664 of 25,361 training images).

But perhaps the most interesting aspect of the training set is the number of images per label. There is a notable shortage of training images – most labels have only one or two images. The following is the histogram of image counts (ignoring unlabeled images) in regular scale and log scale:





About 41% of the labels (2,073 of 5,004) have only one image, and 25% (1,285 of 5,004) have only 2 images. These two groups together constitute 66% of the labels, making this a special case of an image recognition problem, with little supervised information.

The remaining labels have between 3 and 73 images per label. An imbalanced dataset such as this can lead a standard model to develop a bias toward the labels with a higher number of images.

Algorithms and Techniques

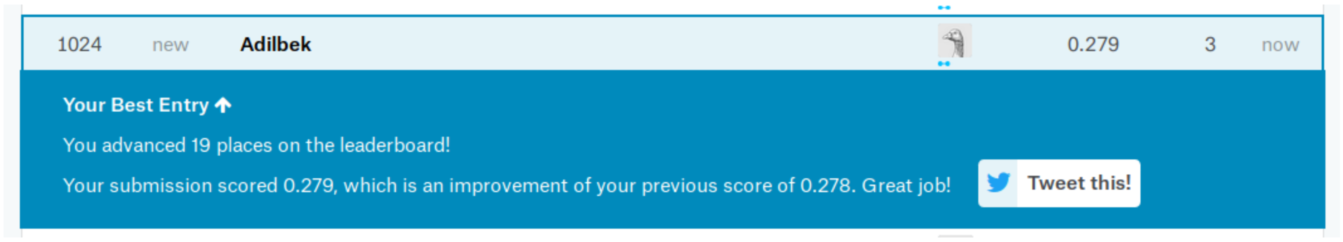
1. Standard CNN

(See notebook “model_1_cnn_basic.ipynb” for implementation details)

The first algorithm I applied to this problem was a standard convolutional neural network (CNN). I constructed this CNN in keras with 3 convolutional layers, each followed by a max pooling layer, and completed with a dense layer. The following is the summary of this model:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 100, 100, 16)      208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 50, 50, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 50, 50, 32)        2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 25, 25, 64)        8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               4608500
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 5004)              2507004
=================================================================
Total params: 7,126,048
Trainable params: 7,126,048
Non-trainable params: 0
```

This constructed model was simple but still powerful – using only minimal image preprocessing where I basically reduced all training images to 100x100 pixels, and ran the training for only 10 epochs (each epoch running less than 2 minutes on CPU), it was able to achieve a decent competition score of 0.279, which ranked the submission in the 76th percentile among all participants:


1024 new Adilbek 0.279 3 now

Your Best Entry ↑

You advanced 19 places on the leaderboard!

Your submission scored 0.279, which is an improvement of your previous score of 0.278. Great job!

I used this model as a baseline/benchmark to compare further modifications and improvements.

2. CNN with image augmentation

(See notebook “model_2_cnn_augmentation.ipynb” for implementation details)

The second model I applied was essentially the same CNN but using image augmentation and some dataset rebalancing. I performed the image augmentation in keras using the *ImageDataGenerator* object and generated new images by performing random rotations, shifts, and flips. The following is an example of an original image and 2 new (augmented) images:

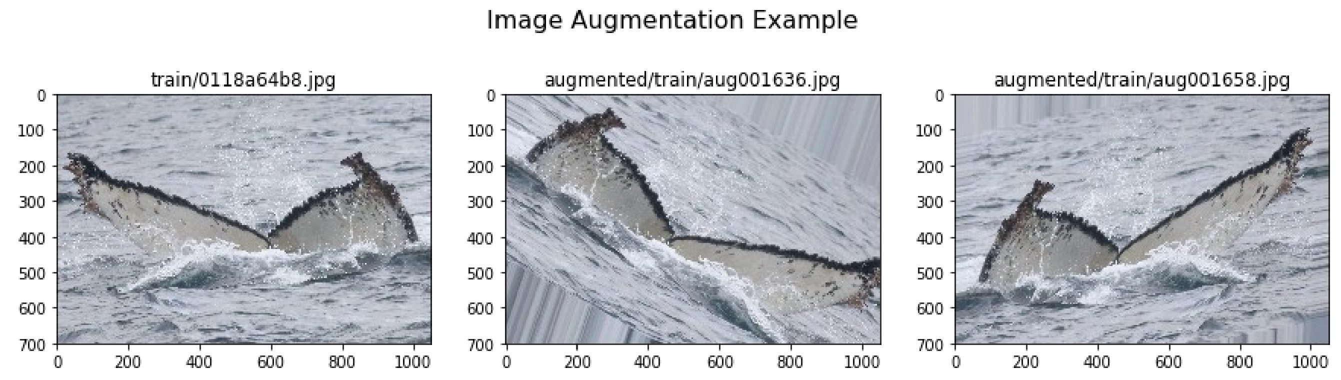


Image Augmentation Example
train/0118a64b8.jpg
augmented/train/aug001636.jpg
augmented/train/aug001658.jpg

Using the augmented images, I rebalanced the dataset to have exactly 5 images per label (for a total of 25,020 images). I used augmented images where there were not enough original images, and dropped some original images where there were more than 5.

The performance of this model was insignificantly worse than the benchmark model's results. The likely reason for its performance is insufficient training images per label, i.e. only 5 images per label (the reason I did not try more than 5 images per label was the memory and disk space issues).

3. Transfer learning

(See notebook "model_3_transfer_learning.ipynb" for implementation details)

For the third model, I implemented transfer learning using the ResNet50 pretrained network. Using keras, I loaded the pretrained network by removing its final layer and retaining the *imagenet* weights:

```
model_resnet50 = ResNet50(weights='imagenet', include_top=False)
```

I obtained and saved the bottleneck features by feeding all preprocessed training images through it and capturing the outputs. Then, I trained only the last layer.

This model did not perform better than the benchmark model either, with the likely reason being the same as earlier – insufficient training images per label.

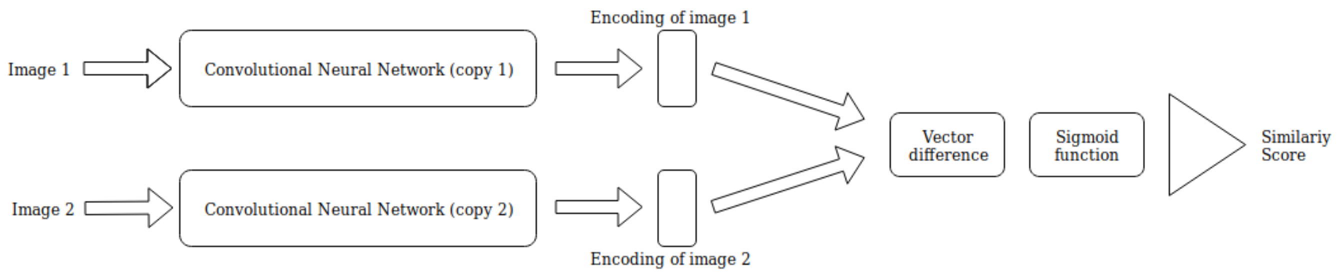
(I have also tried using this model with the augmented and rebalanced dataset from #2, but the score still did not improve significantly)

4. Siamese neural network

(See notebook "model_7_siamese_network.ipynb" for implementation details)

For the final model, I constructed a siamese neural net. A siamese net is an algorithm that is drastically different from even some of the more advanced CNNs, yet it makes use of CNNs in the following way.

It consists of two “twin” CNNs that share the same weights, so the model expects two images simultaneously as input. The two branches are joined at the end, where a simple euclidean distance is computed between the two vectors coming from the branches. Thus, the task of the final layer(s) becomes a binary classification problem (as opposed to categorical classification), deciding whether the two input images are similar or dissimilar:



The unique structure of a siamese network also requires different preprocessing of the training images. The input images must be grouped into pairs, where half of the training pairs must be pairs of same-label images (positive examples), and the other half – different-label images (negative examples).

Siamese networks are especially good for “one-shot” image recognition problems, where very few examples of a target class exist. Given the characteristics of this competition’s dataset, a siamese network is potentially the best candidate algorithm for improving on the benchmark model.

Benchmark

The benchmark model for this project is the first CNN as I described in **Algorithms and Techniques**. This model resulted in the competition score of 0.279, ranking in the 76th percentile among all participants at the time of submission (January 18, 2019).

Methodology

Data Preprocessing

I performed multiple steps of data processing as I progressed through the models. The following is the summary of the early preprocessing steps I undertook:

1. Removed images labeled “new_whale”. Unlabeled images are not useful in training, so removing them simplified the training process.

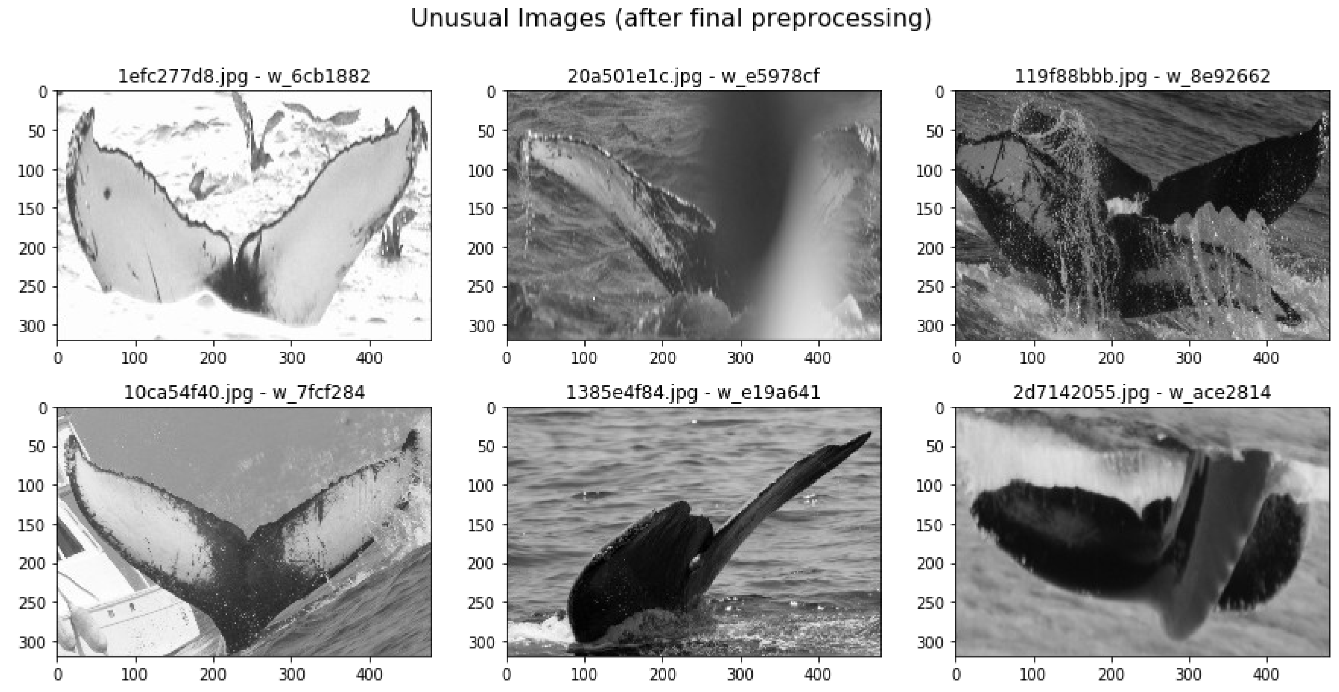
2. Converted the remaining training images to tensors of size 100x100x3. Because CNNs expect numerical inputs, I converted the training images to 3-dimensional arrays (height X width X color channels) and rescaled pixel values to a number between 0 and 1. I implemented this using a custom `img_to_tensor()` function.

3. Generated new images and removed some “extra images”. Using image augmentation in keras, I created new images by applying random rotation, shifting, and flipping. The number of images to

generate was determined based on the goal of obtaining 100 images per label (although due to memory and bandwidth difficulties, I was only able to utilize 5 images per label during training).

4. I created bottleneck features for ResNet50 pretrained network. This process involved creating standard tensors of shape (224, 224, 3) like the *imagenet* dataset originally used to train ResNet50, then running these tensors through the ResNet50 model with its final layer removed to produce tensors of shape (7, 7, 2048).

I performed the most significant preprocessing while preparing the data for a siamese network. During this step, I built a custom python application (`preprocessing_tools/reviewpics.py`) to assist with manually (and efficiently) scanning all images and cleaning them as needed by cropping, rotating, and most notably, flagging unusual images for removal or additional preprocessing later. Here are some examples of such unusual images I found during this process:


Unusual Images (after final preprocessing)
1efc277d8.jpg - w_6cb1882
20a501e1c.jpg - w_e5978cf
119f88bbb.jpg - w_8e92662
10ca54f40.jpg - w_7fcf284
1385e4f84.jpg - w_e19a641
2d7142055.jpg - w_ace2814

These images are unusual because they include objects like birds or boats, were shot at a sharp angle, contain more than one whale, too dark or too bright, etc. and they are likely to skew the training process. Altogether, I marked 450 images as outliers and excluded them from the training going forward.

As part of this final preprocessing, I converted all images to grayscale and resized to 320x480 pixels. The motivation behind grayscale was the fact that some 10% of the images were already in grayscale, and also to reduce tensor dimensions for easier computation. And the reason for selecting the size with larger width than height was to preserve more information – most whale tails are elongated horizontally.

Finally, I selected a random subset of images and created image pairs for input into a siamese network. Here, it was important to ensure that no image was over- or under- represented in the selection, and that there were equal number of positive (same-label) and negative (different-label) examples.

I started by randomly selecting 400 labels with two images per label (for a total of 800 images), and subsequently rearranged them into 800 pairs: 400 positive and 400 negative. In this set, each image appears exactly twice – once as part of a positive example and once as part of a negative example. I used 3/4 of this set for training, reserving the remainder for testing.

Implementation

(See notebook “final_model.ipynb” for implementation details)

The implementation of a siamese network involved the following steps:

1) Preprocess training and testing images as detailed in the previous section.
2) Create tensors from the preprocessed images for model input. The shape of the final tensor should be: [(n, 320, 480, 1), (n, 320, 480, 1)] So, really an array of two tensors where n is the number of image pairs. Min-max normalizing is helpful.
3) Create the branch model with the following structure:
   - Conv2D(filters=64, kernel_size=2, padding='same', activation='relu')
   - MaxPooling2D((2,2),strides=(2,2))
   - Conv2D(filters=128, kernel_size=2, padding='same', activation='relu')
   - MaxPooling2D((2,2),strides=(2,2))
   - Conv2D(filters=256, kernel_size=2, padding='same', activation='relu')
   - MaxPooling2D((2,2),strides=(2,2))
   - GlobalMaxPooling2D
   - Dense(512, activation='sigmoid')
4) Duplicate the branch model for two inputs, both accepting a tensor of shape (320, 480, 1)
5) Create the head model:
   - Add a custom layer to compute the absolute difference between two tensors
   - Add a Dense layer with output 1 and the sigmoid activation function
6) Combine the branch and head models into a single model

   (Optionally, adding L2 regularization to all or some of the layers appears to mitigate overfitting but may also lead the model to never converge)
7) Train the model, monitoring binary crossentropy loss and binary accuracy rate.

   Use dynamically declining learning rate that starts with a default value (0.001 for Adam) and gradually declines by about 90% by epoch 100.
8) Test the model on unseen data, if possible.

Constructing this model was challenging because Keras does not provide a ready siamese model, so it had to be put together more manually, leaving room for error. Please see notebook “model_7_siamese_network.ipynb” for details.

Refinement

In researching siamese networks, I found that they are sensitive to their initial weights. For example, Gregory Koch et al. (on page 10) proposes a specific way to initialize the weights. I have tried initializing them similarly but that lead to an even worse performance. One possible explanation is the difference in the datasets used – the paper is based on the Omniglot dataset which is a relatively smaller and cleaner dataset than the one used in this competition.

Initializing the weights randomly is acceptable but carries a higher risk of the model never converging [7]. In this case, the way the training set is constructed and the order in which the input images are given to the model becomes very important.

Very recently, I found an indication that overfitting may be somewhat mitigated by applying a variable learning rate and L2 regularization. So far, I have used a constant learning rate (either high or low), but some research points to benefits of using a variable rate that declines with some schedule. L2 regularization should help with overfitting too because it assigns a penalty to a more complicated model. Currently, I am still experimenting with various L2 regularization and learning rates by trial-and-error.

Results

Model Evaluation and Validation

(See notebook "final_model.ipynb" for implementation details)

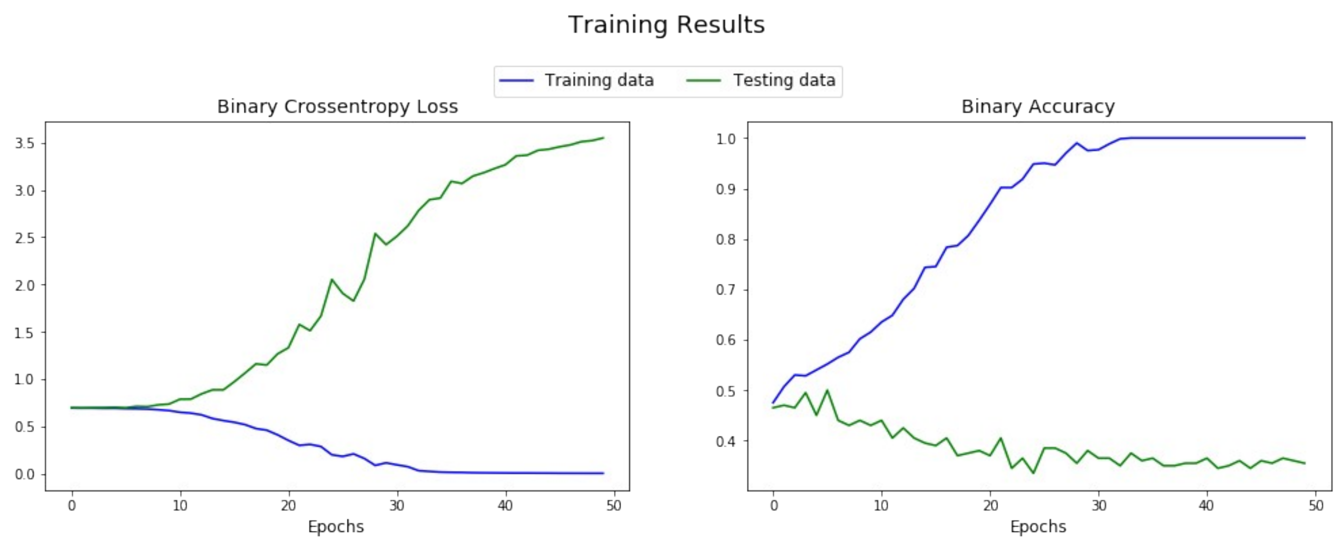
My final model's specifications were as follows:

Version 1:

- The same architecture as described in the **Implementation** section
- Set the learning rate to decline on the following schedule:

```
epochs  1-10:         0.001
epochs 10-30:         0.0008
epochs 30-50:         0.0006
epochs 50-70:         0.0004
epochs 70 and above:  0.0002
```

The following are the results after training for 50 epochs:

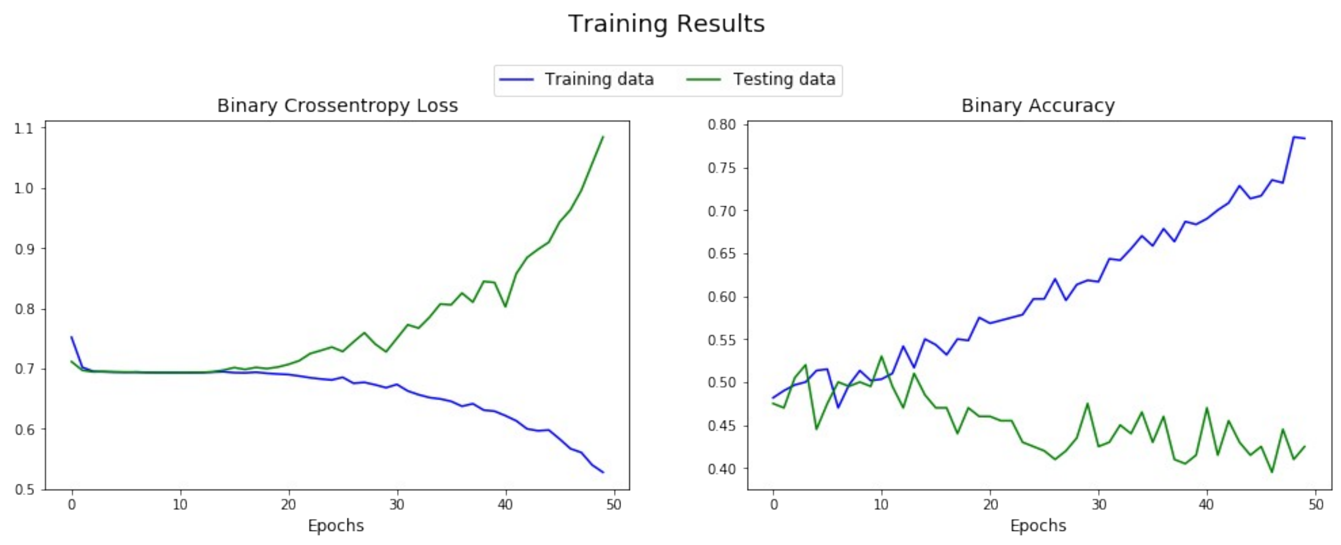


As can be seen from the graphs, the performance on the training and testing sets diverged very early and deteriorated going forward. While the model continued to improve on the training images, the accuracy and loss functions continued to deteriorate. This is likely to be an indication of extreme overfitting.

Version 2:

- Same as version 1, and
- Add L2 regularization with factor 0.0005 in all 3 convolutional layers
- (Experiment with adding Dropout layer with rate 0.2 before the final Dense layer)

The following are the results after training for 50 epochs:

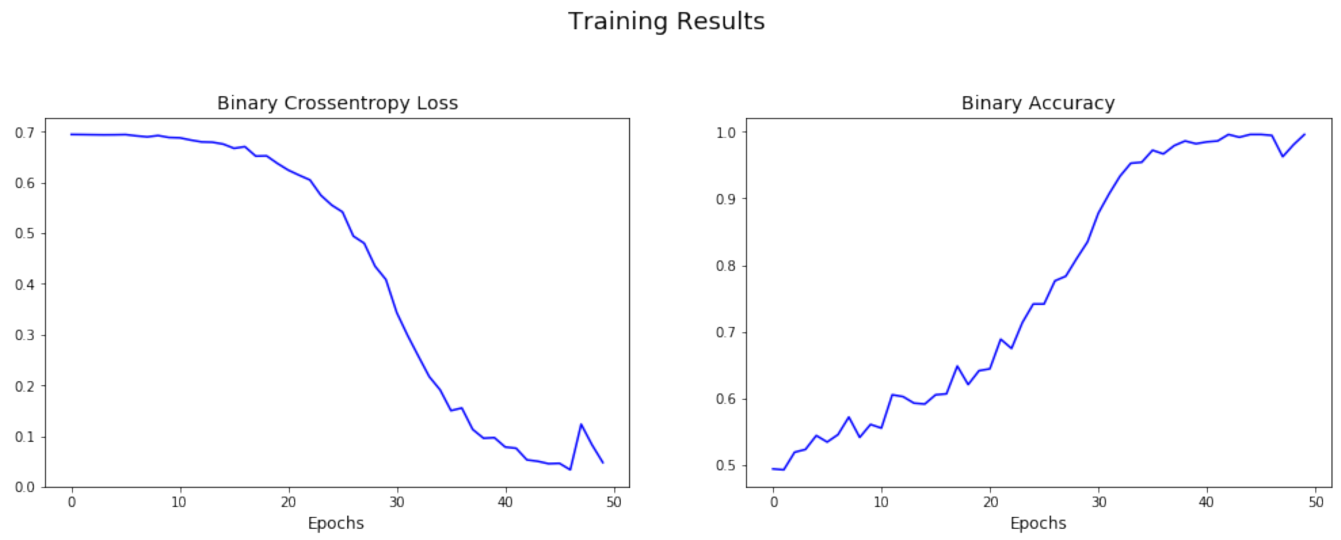


Here, the metrics are still diverging but at a slower rate. The L2 regularization appears to be helping with the problem of overfitting. However, the fact that the testing accuracy never really goes over 0.5 is discouraging (at 0.5 the model is as good as a random model).

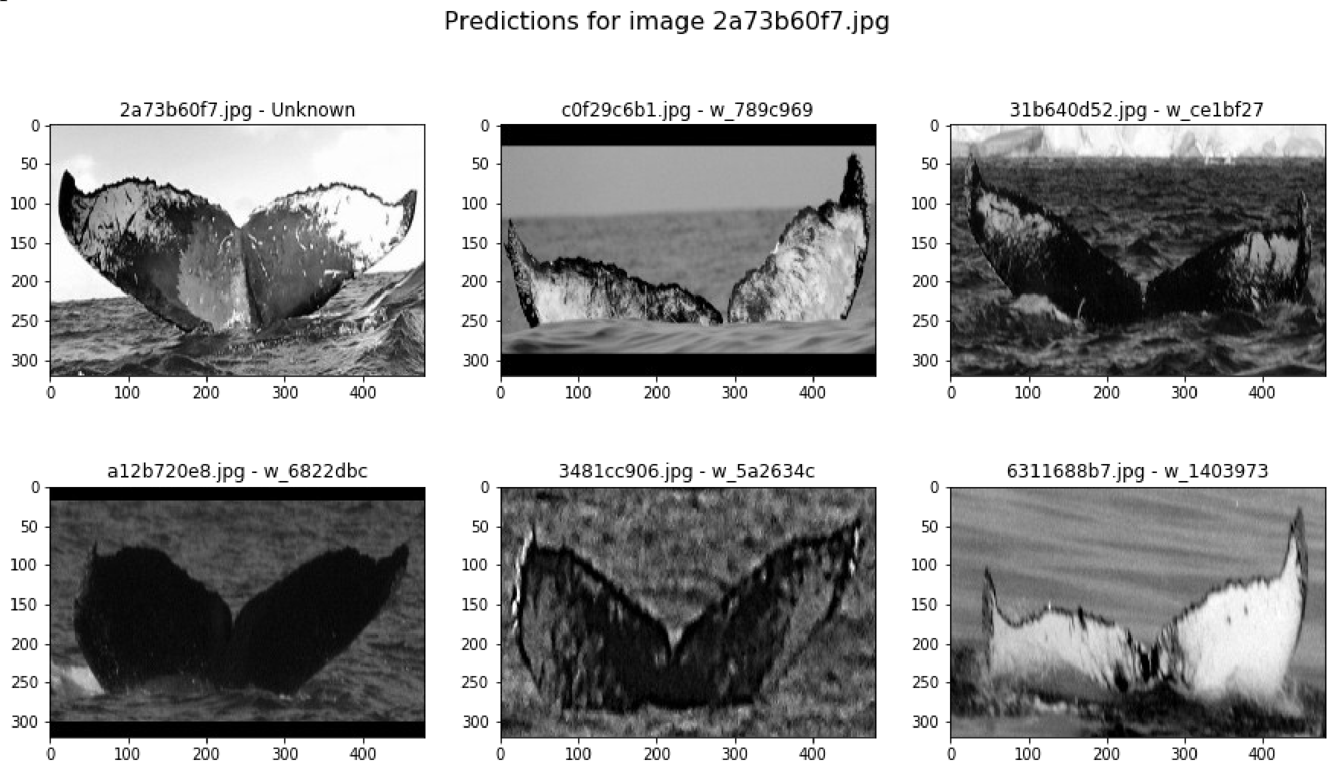
Version 3:

- Same as version 1, and
- Train on all available images (training and validation sets combined)

The following are the results after training for 50 epochs:



Because I used up all labeled images, I decided to test this model by generating predictions on a few actual test images (for which true labels are not known). The following is an example of those predictions:


Predictions for image 2a73b60f7.jpg
2a73b60f7.jpg - Unknown
c0f29c6b1.jpg - w_789c969
31b640d52.jpg - w_ce1bf27
a12b720e8.jpg - w_6822dbc
3481cc906.jpg - w_5a2634c
6311688b7.jpg - w_1403973

The model predicted with very high certainty (over 0.99%) that the whale in the test image (in top left) was similar to the whales in the other 5 images shown above. However, it is clear that most of the 5 predictions are not correct (the bottom left image with whale “w_6822dbc” has a similar shape, so could be counted as a good prediction).

I did not make full predictions for submission to Kaggle for evaluation because I do not believe that my current model can produce a good score based on my above analysis, and because the prediction step is computationally long. In the Kaggle kernel I used for this part, the code to generate predictions for 1 test image took about 5 minutes.

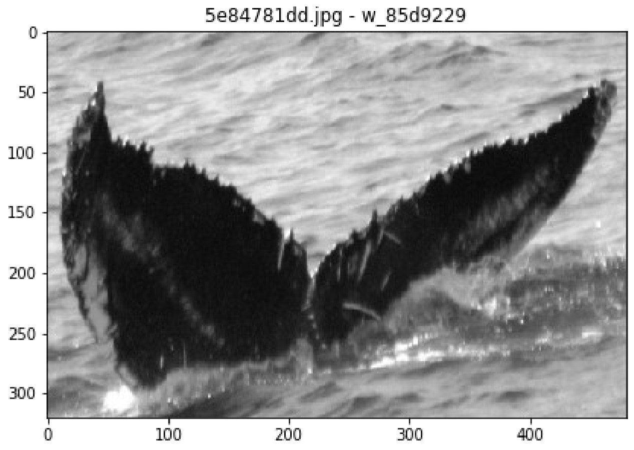
Justification

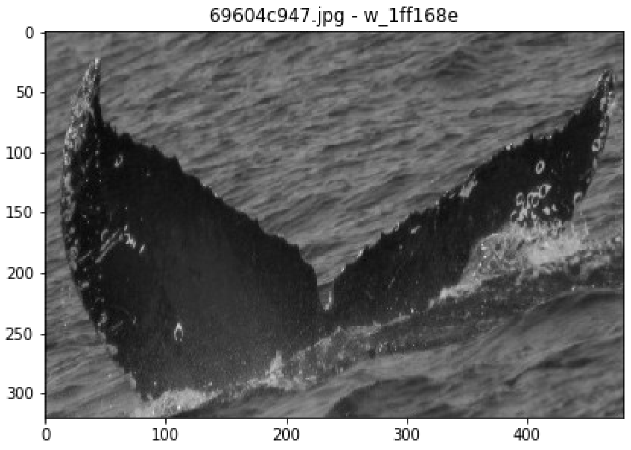
The results of the benchmark model remain my best results overall. The final model was not yet able to improve from the benchmark CNN.

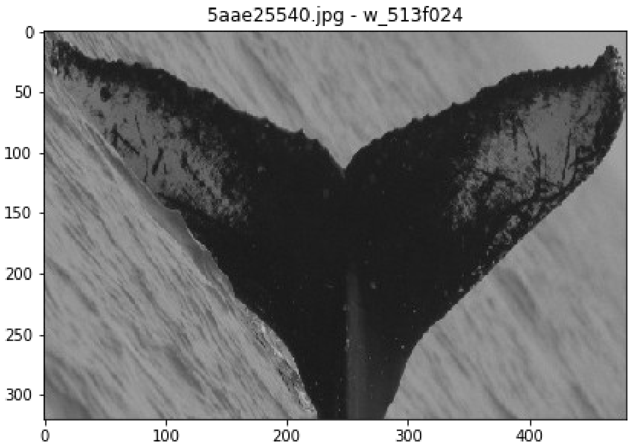
Conclusion

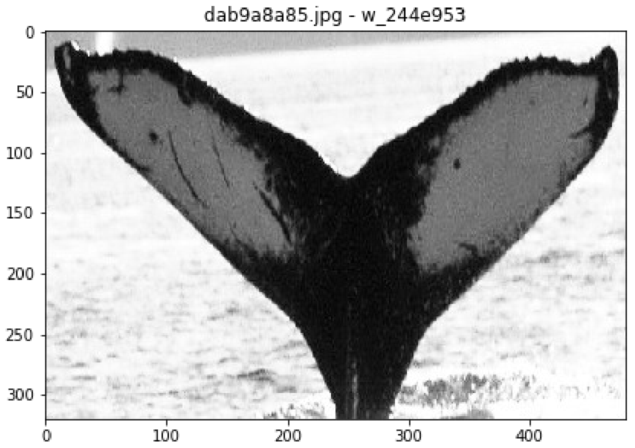
Free-Form Visualization

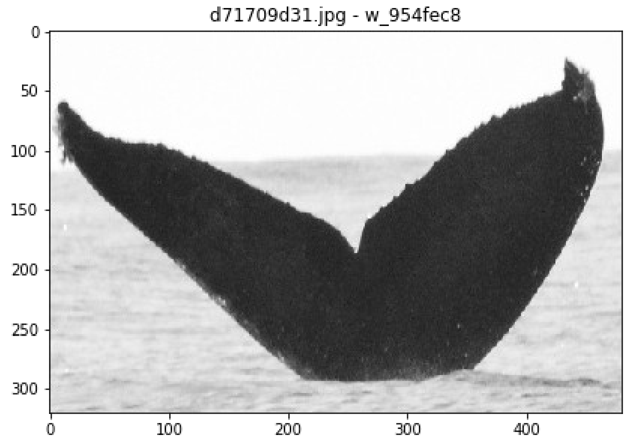
When analyzing images for differences using perceptual hash, or *phash* [8], values I discovered how similar some of the images can appear even to a human eye, despite being images of different whales:

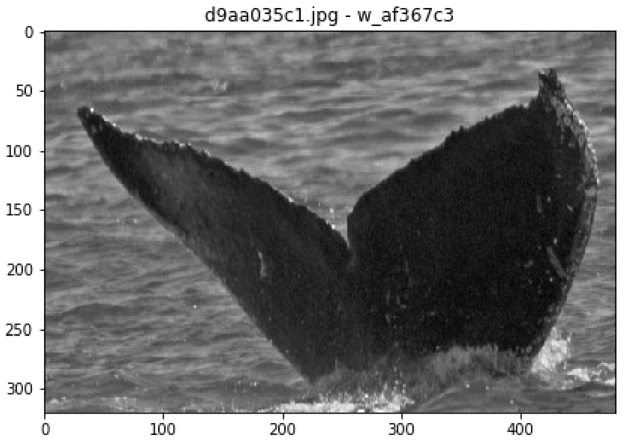






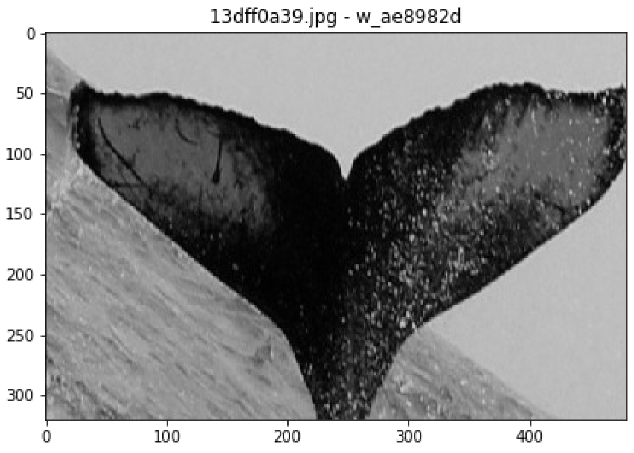


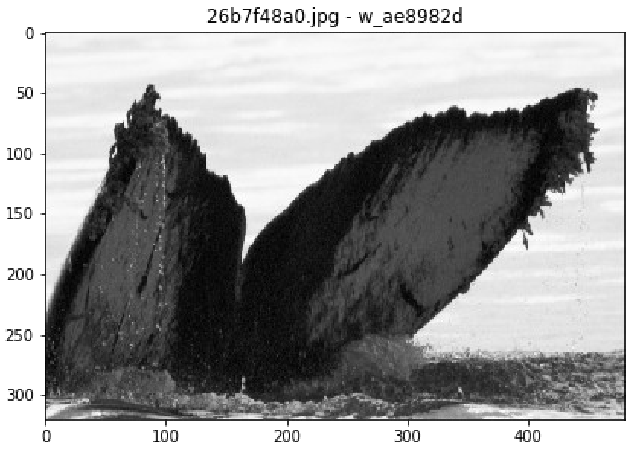




These images show how challenging it may be for the model to train to recognize the relevant features given these obvious yet misleading similarities in features.

The opposite examples also exist where the same whale can appear very differently in two images. Here is an example:





To solve this problem successfully, an algorithm has to learn to ignore the more obvious features like the overall shape and color, and instead look for less obvious ones like small marks and edges. This is very challenging even with a fair number of training examples.

This is yet another reason to try an advance algorithm like siamese neural network in this challenge.

Reflection

The entire process I used for this capstone project can be summarized in the following steps:

1) Dataset exploration, where I identified interesting aspects and challenges in the given data.
2) Data preprocessing, where I prepared the data for use in learning algorithms.
3) Constructing a benchmark CNN model and evaluating its results.
4) Attempting to improve on the benchmark model with image augmentation and rebalancing.
5) Attempting to improve on the benchmark model with transfer learning.
6) Further data preprocessing for a siamese network.
7) And finally, attempting to improve on the benchmark model using a siamese network.

I found the data exploration step fun and useful, and here I improved on my matplotlib skills as well as on pandas and numpy.

I also found the concept of one-shot learning both very difficult and interesting. This topic was completely new to me before I started working on the project, so the learning curve was steep. I found siamese networks particularly interesting because they can effectively learn like humans, i.e. based on very few examples. They bring artificial intelligence a step closer to human intelligence.

Improvement

One area of improvement could be the final model of the twin networks, also known as the ‘head’ model. This model/layer takes the output vectors from the two branch models and compares them to decide if the two input images are of the same class or different. I used a straightforward euclidean distance for this model, but I suspect that this stage can be improved because I can imagine a scenario with two small features and two large features both being different by the same absolute distance. In this situation, a simple distance layer will rank both as equally different, ignoring the difference in the feature sizes. It would be better if the model could factor the size of the features in addition to their absolute difference.

Citations

[1] https://www.kaggle.com/c/humpback-whale-identification
[2] https://www.kaggle.com/martinpiotte/whale-recognition-model-with-score-0-78563
[3] https://en.wikipedia.org/wiki/Siamese_network
[4] http://www.cs.utoronto.ca/~gkoch/files/msc-thesis.pdf
[5] https://www.kaggle.com/pestipeti/explanation-of-map5-scoring-metric
[6] https://www.kaggle.com/c/humpback-whale-identification/data
[7] https://www.kaggle.com/martinpiotte/whale-recognition-model-with-score-0-78563
[8] http://www.phash.org/