Final Report: Automating the Identification of Dophins and Whales

Kaitlin Hutton
Electrical and Computer Engineering
Western University
London, ON, Canada
khutto@uwo.ca

Delaney Stevens
Electrical and Computer Engineering
Western University
London, ON, Canada
dsteve47@uwo.ca

***Abstract*— Happywhale has sponsored a Kaggle competition for a machine learning model that will take images and identify whales and dolphins to assist in marine life research. Datasets containing test and training images, as well as a training.csv file containing the species and IDs of the training images, have been provided. To determine which model would be best, research studies and a previous competition on topics like this project were analyzed, and a Convolutional Neural Network (CNN) model was chosen. The training dataset was explored to determine how the training data was distributed across the various classes and species of marine mammals. Image augmentation was applied during pre-processing to combat dataset inequalities, with the goal of limiting overfitting in the model. A single level CNN was developed, with computing resource constraints limiting the architecture. A validation set was made from the training set, and the model was trained. Hyperparameter tuning was performed. The models' predictions were submitted to Kaggle, and a MAP@5 score was returned. The code and models are available at:**

https://bitbucket.org/katieh010/mlfinalproject/src/main/

Keywords—Machine Learning, Neural Network, whale, dolphin, image classification

I. INTRODUCTION

The tracking of marine mammals, such as dolphins and whales, is an important part of ocean conservation efforts. The location and continued detection of these animals can provide information on population status as well as other trends that can influence research decisions. Currently, tracking marine mammal life requires researchers to wade through vast collections of pictures and manually match/identify each individual mammal with the human eye. As some traits are less prominent and may change over time, this procedure can take "thousands of hours," limiting the scope and accuracy of marine life research. This project aims to solve this problem through the automation of this process by developing a fast and accurate model that can identify individual whales and dolphins based on their natural markings. The classifier will allow researchers to input an image of a dolphin or whale and receive as an output of the top 5 possible individual IDs of the mammal. If the image does not match an existing ID, a new one will be created.

II. RELATED WORK

A. Related Studies

There are many different methods that have been used for Image Classification. In one study on identifying summer crops, different Machine Learning methods were approached. Logistic Regression, Support Vector Machines (SVM), Multilayer Perceptron (MLP) neural networks, and Decision Trees were tested as both single level and hierarchical classifiers [1].

The study looked at four classifiers as flat level classifiers as well as in combination as hierarchical classifiers. For a problem where data may need to be classified into groups and subgroups beyond its final identifiable class, a hierarchical classifying method can be used. Classifiers can be developed for each more general group of data, and the outcome is passed on such that non-viable classes are not included in sub-classifiers. An example of hierarchical classification for this project is shown in Fig.1. below. Note that in Fig.1., the final level of classifiers to get the individual ID has not been included because of the number of IDs.

The study found that a hierarchal classifier performed as well or better than flat classifiers, when using neural networks. However, due to limitations in available computing resources it was determined that a hierarchal system would not be feasible for this project, and a flat level neural network was chosen for the type of model.

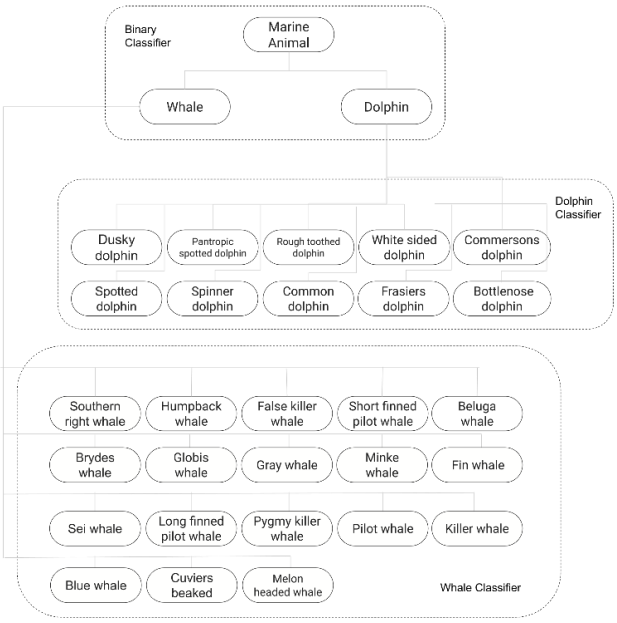


Fig. 1. Hierarchal Classification of the Dataset

B. Previous Competitions

Right Whale Recognition, a competition held in 2016, challenged teams to identify 447 distinct North Atlantic Right Whales in aerial images [2]. It had a smaller data set than Happywhale, with only 4,544 training images available, and focused specifically on one species. The winning team, deepsense.ai, built a model that used a series of multiple Convolutional Neural Networks (CNN) to determine the

identity of each whale [3], [4]. First, a CNN was used to localize the whale's head by creating a bounding box around the whale's head. However, to train the model, the training data had to be manually labeled with the coordinates of the bounding box, which would be challenging with a much larger dataset. After this model identified the location of the head, another model aligned it by identifying the coordinates of the blowhead and bonnet-tip. As all the images had a similar orientation and structure, the main architecture was able to identify the whales more easily. To prevent overfitting, the networks also included supplementary targets that enforced more focus on the important areas of the images (such as the whale's head).

III. Dataset Description

The dataset was acquired from the "Happywhale - Whale and Dolphin Identification" Kaggle competition [5]. It includes a testing folder with 27,000 whale/dolphin images, a training folder with 51,000 whale/dolphin images, and a train.csv file with the species and manually determined "individual ID" belonging to each training image. The test data contains some images of dolphins/whales that are not in the training data and will need to be given a new identification. An example of images provided in the training data set can be seen in Fig. 2. below.

Fig. 2. Example of random images from the training data set. Each image is labeled with their individual ID, species

IV. Methodology

A. Exploratory Analysis

There was a significant class imbalance between the number of images per species in the training data (Fig. 3.). As seen in plot, the most frequent species (the Bottlenose Dolphin) has 10,781 images, 2,000 more than the next most common species and 770 times more than the least common species (the Fraisers Dolphin). This meant that the original data required augmentation to prevent bias towards the Bottlenose Dolphin and poor classification of the Fraisers Dolphin.

There was also an imbalance in the number of individual IDs per species (Fig. 4). While some species accounted for the majority of individual IDs, they did not account for the majority of images, and vice versa. For example, the dusky dolphin species in Fig. 4 (teal bar) makes up a large number of individual IDs, however it only accounts for a small portion of the dataset as shown by the teal bar in Fig. 3.

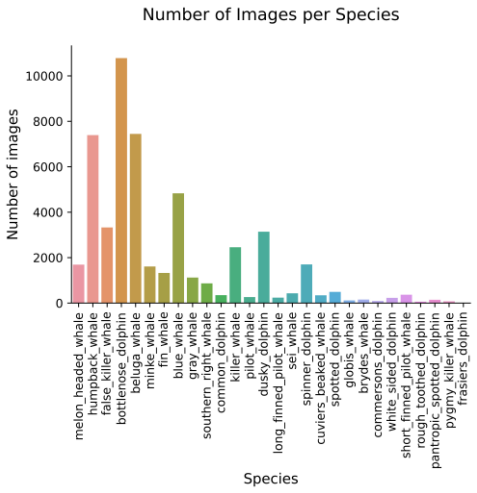


Fig. 3. Distribution of Images per species

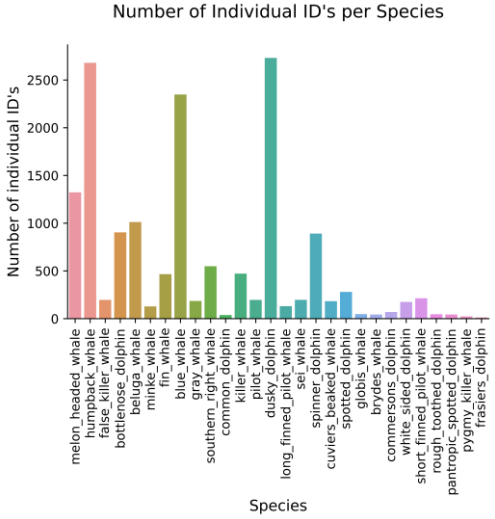


Fig. 4. Distribution of animals recorded per species

The training data was also found to have a large class imbalance between individual animal IDs, as well as a significant lack of data for a majority of IDs. As shown in Fig. 5. below, approximately 59% of IDs have only one image in the training dataset while the most frequent ID has 400 images. This further suggested that the original data required augmentation to ensure enough data is available for training and to prevent bias towards the ID with 400 images.

```
Number of IDs 1 or fewer training images: 9258
Percentage with 1 or fewer training images: 0.5939565022133829
```

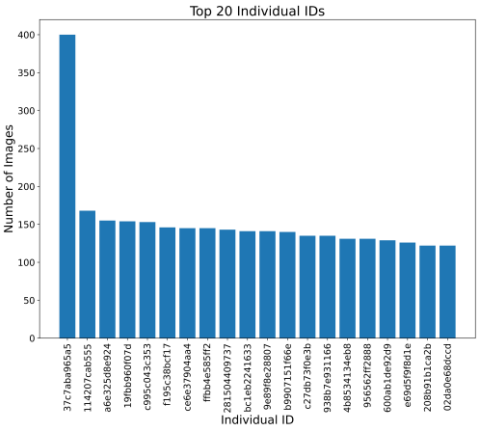


Fig. 5. Distribution of images for the 20 most photographed individual IDs

The training images were also found to have varying image sizes and aspect ratios as shown by the first 5000 images in the dataset (Fig. 6 and Fig. 7). When running a model that requires consistent image sizes, such as a fully connected model, this can be a problem. Two options were considered to cope with this. The first was to scale the images to the same size while maintaining the most common aspect ratio, which is 0.7 in Fig. 7. The second alternative was to use a Fully Convolutional neural network as it has no limitations on the input size [6]. It was decided that scaling the images would be used due to its simplicity.

Fig. 6. Scatterplot depicting the height and width of the first 5000 images in pixels

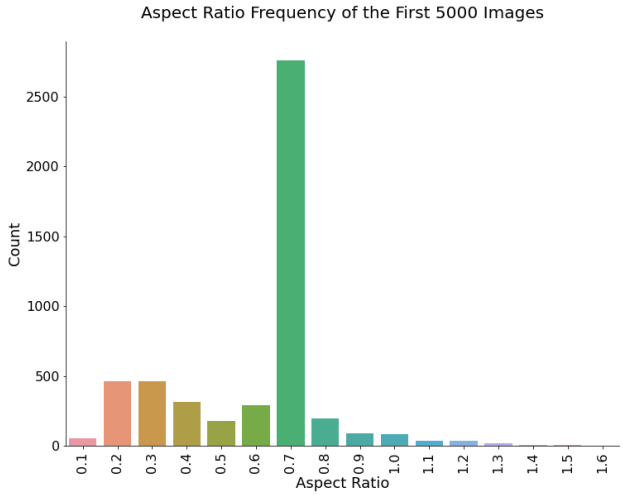


Fig. 7. The aspect ratio frequency of the first 5000 images

The code used to perform this analysis is contained in the Exploratory_Analysis.ipynb file in the repository.

B. *Preprocessing*

To compensate for data imbalances, data augmentation was performed using the Keras ImageDataGenerator [7]. ImageDataGenerator alters images by changing their rotation, brightness, axis, zoom focus, and other properties (Fig.8.). This guarantees that the model is never trained on the same image twice, which can prevent overfitting. It also makes a more robust model by creating more data for the classes where there is only one image. The acceptable ranges for the data augmentation were chosen to provide a diverse range of augmented images, while still remaining reasonable for what expected data would look like.

```
augDatagen = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.20,
    brightness_range=[0.7,1.0],
    channel_shift_range=0.7,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    fill_mode='nearest'
)
```

Fig. 8. Data Augmentation Parameters

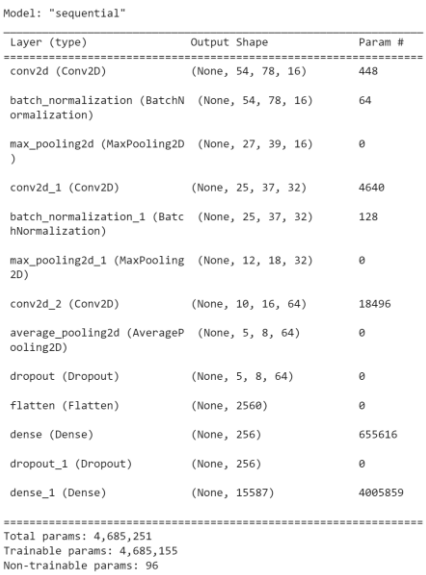
The first attempt at loading augmented images involved the use of ImageDataGenerator's built-in function, flow_from_dataframe. The function would get the paths of each image directly from the data frame and augment and load in real time, while the model was fitting. This proved to be a problem because the available computers lacked sufficient RAM to do the task in a timely manner. Even when using a GPU this method caused a bottleneck because the images were being loaded by the CPU before being transferred to the GPU for fitting to the Keras model. When discovering this, a different method was used.

The second method still involved using ImageDataGenerator, however all the necessary images were augmented ahead of model fitting. The images were passed through the data generator and saved to a separate generated train images folder using an augImages (Data_Augmentation.ipynb) function. Only the IDs with a low number of images were augmented, so that unique images took precedence over augmented ones. This method enables all images to be loaded and converted to an array in advance of model fitting (CNN_w/_data_augmentation.ipynb). While loading the images still took a few hours, fitting the model was much faster, and multiple models could be tested on the pre-loaded images.

Unfortunately, even with this method, only three images could be loaded per individual ID. This was due to RAM limitations. Even with only three images per ID, there were 46,761 training images, and loading more resulted in the runtime crashing. To maintain the desired aspect ratio of 0.7, the images were loaded into a size of 56x80 (height x width) pixels. Attempts were made with a larger resolution, but this caused additional issues with RAM. More of this is discussed in the Challenges and Future Considerations section of the report.

C. *Model & Architecture*

Based on the previous research, CNN models were determined to be the best model for this image classification problem. The Keras Sequential model was used to create a basic, flat-level classifier (Fig. 9). The model's architecture was based on deepsense.ai's main model developed for the Right Whale Recognition competition [2]. The model was built following the standard practice for CNNs, where a convolutional layer is followed by a sub-sampling pooling layer for feature extraction. Due to the low resolution of the images, only three pairs of convolutional layers and pooling layers were used. Each pooling layer reduced the image dimensions by half, resulting in images with a resolution of 7x10 after the third layer. It was thought that adequate features could not be extracted from images smaller than that size.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 54, 78, 16)        448

 batch_normalization (BatchN  (None, 54, 78, 16)       64
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 27, 39, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 25, 37, 32)        4640

 batch_normalization_1 (Batc  (None, 25, 37, 32)       128
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 12, 18, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 10, 16, 64)        18496

 average_pooling2d (AverageP  (None, 5, 8, 64)         0
 ooling2D)

 dropout (Dropout)           (None, 5, 8, 64)          0

 flatten (Flatten)           (None, 2560)              0

 dense (Dense)               (None, 256)               655616

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 15587)             4005859

=================================================================
Total params: 4,685,251
Trainable params: 4,685,155
Non-trainable params: 96
```

Fig. 9. Model summary showing layers and parameter numbers

Batch normalisation layers were implemented twice to help with overfitting. Batch normalisation layers standardise the prior layer's activations and can be used as model regularization. Dropouts were also added to help with overfitting because they introduce randomness to the model. The ReLu activation function was applied to each of the convolution layers as it was determined to be the best activation function for layers of this type. The activation function for the output Dense layer was set to softmax, so that the predictions would be the percent likelihood of an image belonging to a specific class. Using the Scikit-learn LabelEncoder and OneHotEncoder classes, the target data was One Hot Encoded. This allowed an array of numerical binary values corresponding to each output class to be used instead of the given individual id label of type String.

Prior to tuning, the model was compiled using the Adam optimizer with the loss set to categorical cross entropy (Fig. 10). It was fitted on both a training and validation set. Having a validation set can help with overfitting, however the benefit was determined to be insignificant for this project due to the nature of the data. Since only three images could be loaded per class, two were used for the training set and one was used for the validation set. For many of the classes that only had one original image, this meant that their corresponding validation image was an augmented one. All of these factors have a negative impact on the model's accuracy when making predictions.

```
model.compile(loss = 'categorical_crossentropy',
              optimizer = 'adam',
              metrics = ['acc',tf.keras.metrics.AUC()])
```

```
history = model.fit(
       x = Xtrain,
       y = Ytrain,
       epochs = 40,
       batch_size = 128,
       verbose = 1,
       callbacks = callbacks
       )
```

Fig. 10. Compilation and fitting of model

D. Hyperparameter Tuning

Three hyper-parameter optimization methods were considered due to their ability to handle a large hyperparameter space: Bayesian Optimization with Tree Parzen Estimator (BO-TPE), Particle Swarm Optimization (PSO), and Genetic Algorithm (GA). BO-TPE was ultimately chosen as it can detect optimal hyper-parameters in the shortest computational time (time complexity of O(nlogn)) and does not require good initialization [8].

However, these methods required more memory than was available, preventing optimization from being completed. Several solutions were attempted, including reducing the configuration space, using a smaller dataset while increasing the folds, and saving trial data to a pickle file after each trial. However, the optimization still could not be completed. The code that would have been implemented if more RAM had been available was designed using an example from class and is in implemented in the last two cells of the Hyperparameter Optimization_with_BO-TPE.ipynb file in the repository. The hyperparameters and configuration space that would have been tested are shown in Table I. Due to the nature of the model, the other model parameters were limited to one option, so they were not tested (for example, the model is solving a multi-classification problem, so categorical cross entropy was chosen for the loss function).

TABLE I

Hyperparameter Configuration Space used for Optimization	
Hyperparameter	***Space***
Learning Rate	Log uniform from 0.001 to 0.9
Batch Size	Choice between [32, 64, 128, 256]
Epochs	Discrete uniform from 20 to 60 (spaced by 10)
Dropout Rate	Discrete uniform from 0.2 to 0.5 (spaced by 0.1)

Because of these constraints, the parameters were chosen through manual tuning. The same hyperparameters listed above were tested as shown in Table II, but with a much smaller configuration space. Table III shows the parameters that were selected and why they were selected.

TABLE II

Hyperparameter Configuration Space used in Manual Tuning	
Hyperparameter	***Space***
Learning Rate	0.001, 0.01
Batch Size	64, 128, 256
Epochs	20, 30, 40
Dropout Rate	0.2, 0.3, 0.4

TABLE III

Selected Hyperparameters		
Hyperparameter	***Chose***	***Reason***
Learning Rate	0.001	Only rate that achieved convergence
Batch Size	128	64: loss didn't change, 256: bad accuracy
Epochs	40	Good training accuracy without crashing
Dropout Rate	0.4	Training loss increased on 0.2,0.3

E. *Top 5 Predictions*

To meet the requirements of the Happywhale competition, the top five individual ID predictions for each of the test images were required. Because of the output layer softmax activation function, the percent likelihood of an image belonging to a specific class was known, making it easy to get the top five predicted classes. However, Happywhale added a new requirement for the predictions. Some of the images in the testing set were entirely new dolphins/whales that had never been seen before and did not correspond to an ID that the model had been trained on. Happywhale asked that these test images be labelled as new_individual as they would require a new ID. The way this was approached in this project was to look at the prediction percentages made by the model and set a threshold value. If a test image had a low prediction percentage for a class, it was more likely to belong to a new individual than to that class. All top five predictions for an image that were below the threshold percentage were set to the new_individual. The code for the prediction process is in the file Predictions.ipynb.

The threshold was chosen through manual tuning. As a majority of the training dataset contains individual ID's of an animal that was only ever seen once, it can be assumed that most animals are a spotted only a few times. Therefore, it is likely that future sightings will involve new animals that are also only seen a few times, so a large threshold was selected that would ensure more new_individuals are created. Accounting for overfitting and model iaccuracies, the threshold for labeling a test image with new_individual was set to 60%.

V. Results and Analysis

A. *Evaluation Metrics*

The model was evaluated using three metrics: training accuracy, validation accuracy, and the MAP of true and false positives, where each image has five IDs predicted. Ideally , a one vs. rest AUC would have been used to assess the model's ability to differentiate between classes. To do so, however, each of the 15,578 classes would have to be evaluated against the others, which would require more memory than was available. As no ground truth test.csv file was provided, the only way to test the model was to upload the data to Kaggle and retrieve a MAP@5 score.

B. *Results and Analysis*

The model was able to train on the data, as indicated by the blue line in Fig. 11 and Fig. 12. The training loss curve is decreasing over time, while training accuracy is increasing, indicating that the model is learning. However, as illustrated by the orange lines in Fig. 11 and Fig. 12, the validation loss is increasing and accuracy is only slightly increasing before plateauing, indicating that the model is overfitting to the training data. Given more time and resources, solutions such as implementing more regularization methods (such as increasing the dropout rate) or using images with higher resolution to improve feature extraction can be implemented. The model had a total training loss of 0.37. Overall, the model had a training loss of 0.27, a training accuracy of 0.86, a validation loss of 10.92, and a validation accuracy of 0.16.

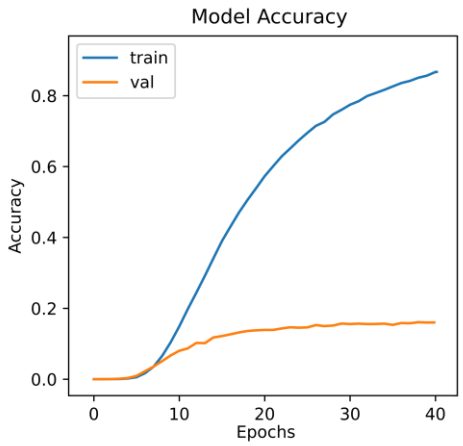


Fig. 11: Training and Validation Accuracy

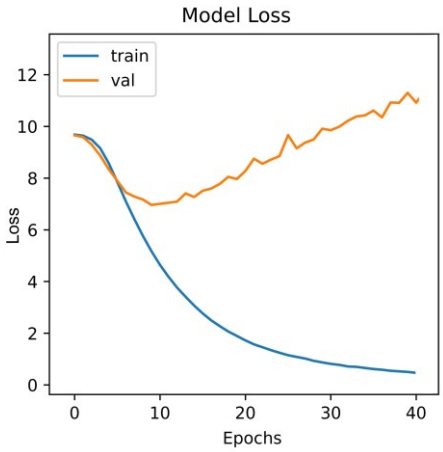


Fig. 12: Training and Validation Loss

The top five predictions made on the test data were passed to the Kaggle Happywhale competition site, where the model received a MAP@5 score of 0.03. A sample of the predictions made and the Kaggle score received are shown in Fig. 13 and Fig. 14, respectively. The MAP@5 score did not meet the result of 0.8 expected in previous reports. However, after reviewing the validation results, this outcome appears to be reasonable. The model's poor performance is most likely the result of a combination of factors. The first was that, due to computational constraints, the model's depth had to be reduced and the dataset images had to be compressed, resulting in poor feature extraction. Second, some images were cropped when loaded to ensure that the image size was identical when passed to the model; as a result, the model was not trained on all features. Third, because there are so few images in each class in the training data, the model is more prone to overfitting. Lastly, the probability threshold for determining whether the image belonged to a new ID was determined manually. It is likely that a higher performing threshold exists, but future tests would be required to determine if this is true.

image	predictions
003c46063bb0c4.jpg	b5eebed8e08b 2d3ed1986ddf c971fb024e55 c05071b0697f new_individual

Fig. 13: Top 5 Predictions sample

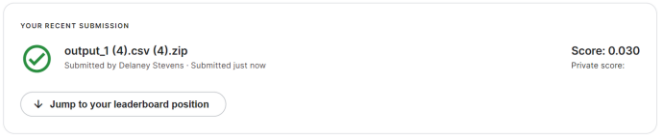


Fig. 14: Kaggle MAP@5 score

VI. Challenges and Future Considerations

A. Challenges

As it has been discussed throughout this report, many challenges arose while trying to implement this project. Many of them stemmed from the lack of computing power available. For example, a big issue was that all the training images needed to be loaded for model fitting, which caused the runtime to crash. This was addressed by using fewer images during training and having those images be of lower resolution. As a result, there was a limit to the depth of the CNN since subsampling more than three times would result in an unusable image resolution.

Additionally, the smaller starting resolution meant that many details on the dolphins and whales were lost. If the problem was to identify the animal from a data pool of different types of animals (cats, dogs, etc.), the shape of the animal in the image would be enough for the model to differentiate what the animal is. However, for this problem the model is trying to determine the specific individual animal. It needs to be able to detect the marks, scars, and particularities which were lost at a low resolution. This model would most likely perform better if run at a higher resolution.

The dataset was also a challenge for this project. Due to the scarcity of images in many of the classes, augmentation was needed to perform any useful training. This, however, makes the model more prone to overfitting. In addition, the size of the training images varied. While resizing each image was used to combat this in this project, doing so crops images that do not fall within that ratio, most likely resulting in information loss and thus low performance on unseen data. Other solutions that can combat the image size variation problem are discussed in Future Considerations.

B. Future Considerations

A major future consideration would be to re-run this project on a computer with more computing power to address the previously listed challenges. With more RAM and perhaps multiple GPUs, the images would be able to be loaded with a higher resolution. More images could be augmented so that the model could train with more images in both the training and validation sets. Additional convolution and pooling layers could be added if that was determined to be beneficial to the model. Implementing one vs. all AUC could be used to properly evaluate the model. Additionally, hyperparameter optimization could be performed to find parameters that would improve validation performance. Furthermore, with more RAM, the hierarchal classification model could be tested to see if it produced a higher MAP@5 score than the single level classifier. If it had been feasible on the current set-up, this approach would have been tried out sooner.

The process for dealing with image size variations needs to be improved. A fully connected network could be implemented in the future because it is unaffected by image size. Another solution would be to zero-pad the images to the same size, which would preserve information and patterns. Furthermore, a study found that when comparing the training time of the zero-padding method to the resizing method, this method is significantly faster [9].

References

[1] J. Peña, P. Gutiérrez, C. Hervás-Martínez, J. Six, R. Plant, and F. López-Granados, "Object-Based Image Classification of Summer Crops with Machine Learning Methods," *Remote Sensing*, vol. 6, no. 6, pp. 5019–5041, May 2014, doi: 10.3390/rs6065019

[2] "Right whale recognition," *Kaggle*. [Online]. Available: https://www.kaggle.com/c/noaa-right-whale-recognition. [Accessed: 05-Mar-2022].

[3] R. Bogucki, M. Cygan, C. B. Khan, M. Klimek, J. K. Milczek, and M. Mucha, "Applying deep learning to right whale photo identification," *Conservation Biology*, vol. 33, no. 3, pp. 676–684, 2018.

[4] R. Bogucki, "Which whale is it, anyway? face recognition for right whales using Deep learning," *deepsense.ai*, 05-Jan-2021. [Online]. Available: https://deepsense.ai/deep-learning-right-whale-recognition-kaggle/. [Accessed: Feb-2022].

[5] "Happywhale - Whale and dolphin identification," *Kaggle*. [Online]. Available: https://www.kaggle.com/c/happy-whale-and-dolphin/overview/description. [Accessed: Feb-2022].

[6] A. Thakur, "How to handle images of different sizes in a convolutional neural network," *W&B*, 19-Aug-2020. [Online]. Available: https://wandb.ai/ayush-thakur/dl-question-bank/reports/How-to-Handle-Images-of-Different-Sizes-in-a-Convolutional-Neural-Network--VmlldzoyMDk3NzQ. [Accessed: 01-Mar-2022].

[7] P. Pai, "Data augmentation techniques in CNN using tensorflow," *Medium*, 03-Nov-2020. [Online]. Available: https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9. [Accessed: 05-Mar-2022].

[8] L. Yang and A. Shami, "On hyperparameter optimization of Machine Learning Algorithms: Theory and practice," *Neurocomputing*, 25-Jul-2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220311693?via%3Dihub. [Accessed: 01-Apr-2022].

[9] M. Hashemi, "Enlarging smaller images before inputting into convolutional neural network: Zero-padding vs. Interpolation - Journal of Big Data," *SpringerOpen*, 14-Nov-2019. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0263-7. [Accessed: 11-Apr-2022].

Contribution Statement

The contribution was divided equally throughout the duration of this project. Report writing, exploratory data analysis, model design research, and implementing model predictions were all discussed together. Katie researched and implemented data preprocessing, while Delaney researched and implemented hyperparameter tuning and optimization. Katie also carried out the model's implementation, while Delaney ran and tested the model.