# Classifying and Detecting Fish Species using Images from Boat Cameras

Adrienne Mok, Bill Li, Carson Hu

***Abstract*—Our project offers a solution to the Kaggle challenge proposed by The Nature Conservancy. We employ an algorithm that automatically classifies and detects fish species into eight different categories using images captured by boat cameras. To solve the multi-labeled classification problem we fine-tune a pre-trained convolutional neural network and classify each image from the testing set into the appropriate category. We evaluate our success based on the multi-class logarithm loss.**

## I. INTRODUCTION

### A. *Motivation*

The Nature Conservancy uses boat cameras to monitor fishing activities, however the sheer amount of data obtained is inconvenient to analyze manually. Thus, developing an algorithm that accurately recognizes and classifies fish species would allow The Nature Conservancy to identify which areas need conservatory attention much more efficiently.

### B. *Definition*

We solve the proposed multi-label classification problem by using pre-trained models based on the convolutional neural network (CNN) framework. CNNs are especially useful for image recognition and classification problems since they produce accurate results for learning image representations. They typically consist of various elements, such as convolutional, pooling, and fully connected layers. We add several of these layers on top of each other to create a deep CNN that can effectively learn different features. We apply fine-tuning to our training dataset and ultimately classify each fish from the test dataset into one of the following eight species categories:

0 - Albacore tuna
1 - Bigeye tuna
2 - Yellowfin tuna
3 - Mahi Mahi
4 - Opah
5 - Sharks
6 - Other
7 - No Fish

### C. *Description of the Dataset*

We use the dataset provided by The Nature Conservancy, which includes training images organized by fish species labels as well as unlabeled testing images. There are eight target categories, including Albacore tuna, Bigeye tuna, Yellowfin tuna, Mahi Mahi, Opah, Sharks, Other, and No Fish. The training folder contains 3777 images, which are divided into one of eight subfolders corresponding to the appropriate target categories. The provided testing folder is comprised of 1,000 unlabeled images. Each image must be classified into exactly one of the eight categories. **Figure 1** shows some sample images from each class.


**FIGURE 1: SAMPLE IMAGES-** Examples of images from each class cropped to dimensions 224 x 224.

The dataset presents several challenges. Other datasets used for CNNs typically contain millions of images, while the provided dataset consists of only several thousand images. The images do not share a common dimension and were often taken under different lighting conditions. In some pictures, humans were blocking the view of the fish, and in many cases, the fish occupied only a small portion of the entire image. Furthermore, fish species are not evenly distributed among the training set, with Albacore tuna occupying the majority of the dataset. Considering these limitations, an algorithm that effectively classifies fish species with reasonable accuracy is necessary.

## II. ALGORITHM DESCRIPTION

### A. *Methods*

Our algorithm solves the multi-label classification problem by fine-tuning a pre-trained convolutional neural network (CNN) using the provided training set. First, we randomly set aside 500 images for validation. Before building our CNN, we pre-process the images. We normalize the dataset by subtracting the mean pixel value from each pixel and use the provided *ImageDataGenerator* function in Keras to augment the training data with various transformations to the images. After pre-processing the images, we build our CNN. The pre-trained CNN is obtained from the deep learning framework Keras using the VGG 16-layer model trained on ImageNet with batch normalization. We perform transfer learning on our CNN model and fine-tune it for optimal results. In this manner, the pre-trained model acts as a feature extractor. To fine-tune our model we remove, or "pop", the last five layers of our CNN and replace it with our own classifier. We then freeze the weights of the other layers and train the network normally. By freezing the other layers, we ensure that the weights do not change during gradient descent. During the testing phase we use the *ImageDataGenerator* function again to augment the test data and create pooled predictions. After the predictions have been made, we submit the predicted probabilities of each class for all the images in the testing set in a .csv file.

## III. EXPERIMENTAL RESULTS

### A. *Evaluation of Success*

The success of our algorithm is evaluated by the multi-class logarithm loss. For each image, a set of predicted probabilities must be submitted. The formula is shown below:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

where N is the number of images in the test set, M is the number of image class labels, $y_{ij}$ is 1 if observation $i$ belongs to class $j$ and 0 otherwise, and $p_{ij}$ is the predicted probability that observation $i$ belongs to class $j$.

### B. *Results from Algorithm*

Initially we classified images solely with a CNN trained from scratch using a typical structure based on other known CNN structures, which resulted in a multi-class logarithm loss score of 1.10 (Top 30% of Kaggle's Leaderboard). However, since the training dataset is not as comprehensive as other datasets typically used to train CNNs from scratch, its size is insufficient for the depth of the network. Thus, in order to obtain more accurate results, we implemented the transfer learning model described in our algorithm. The results acquired from transfer learning were more representative of their true class, and using this model we were able to achieve a multi-class logarithm loss score of 0.95 (Top 11% of Kaggle's Leaderboard). **Figure 2** provides information regarding the logarithm loss and accuracy of our training set. In general, we want to minimize the multi-class logarithm loss score and maximize the level of accuracy.

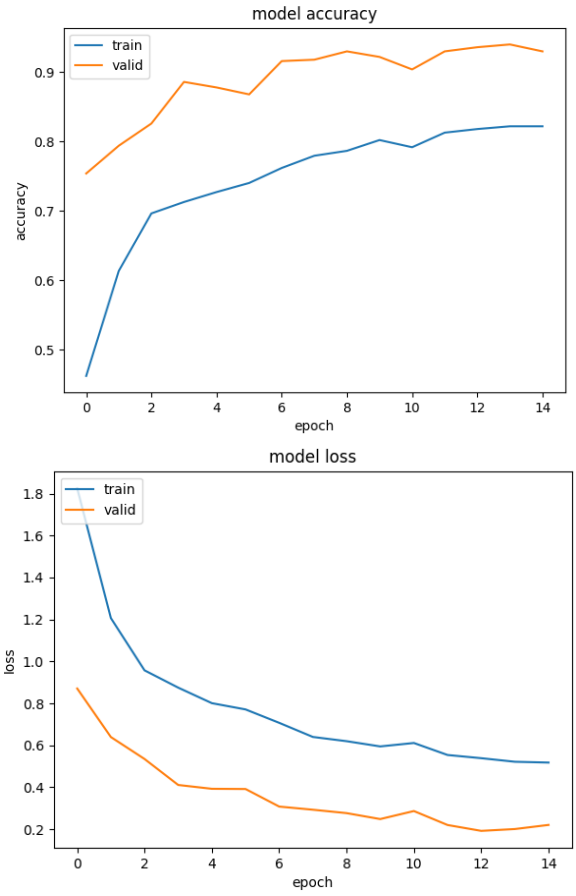


**FIGURE 2: METRICS OF TRAINING SET-** Accuracy increases over the number of epochs. Accuracy is 0.8220 for the training set and 0.9300 for the validation set. Logarithm loss decreases over the number of epochs. Logarithm loss is 0.5177 for the training set and 0.2204 for the validation set.

### C. *Analysis of Datasets*

As mentioned before, the dataset is highly unbalanced, since the provided images of fish species in the training set are not evenly distributed. **Figure 3** provides a view of the distribution. The majority of the dataset is composed of Albacore Tuna, which can inevitably lead to skewed results.

To account for the uneven distribution of the dataset, we propose combining boosting with our CNN model, which emphasizes outliers such as samples which have been incorrectly classified. Boosting trains a new classifier that learns to fix the errors from previous classifiers. We use the weak classifiers from our training set as base learners so that we can formulate different decision boundaries. A new model is then trained with the base learners. We repeat the process multiple times for optimal results. By reducing bias, this method is expected to yield better results and reduce the number of incorrectly classified images.

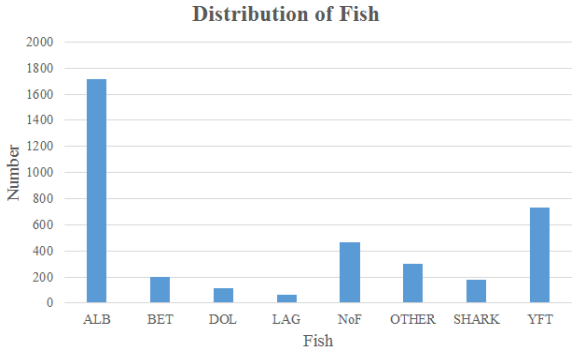


**FIGURE 3: DISTRIBUTION OF DATASET-** Albacore tuna (ALB) occupies the majority of the training dataset.

## IV. CONCLUSION

Because training CNNs from scratch can be a tedious process, using pre-trained CNNs offers a much more practical solution while still maintaining a reasonable level of accuracy. Furthermore, the small size of the dataset renders it insufficient for the depth of the network when training a CNN from scratch. Through transfer learning, we used features trained on the large dataset ImageNet and trained a classifier on top of those features. Ultimately we were able to fine-tune the pre-trained CNN and obtain more accurate results than using a CNN trained from scratch. In the future, we intend to implement boosting to account for the skewed training dataset distribution in order to achieve an even greater representation of the true class value.