CS529 Project 3: Whale Classification

Mike Adams and Jack Ringer

April 2023

Note: All code used for this project can be found at our GitHub repository https://github.com/mikethebos/cs529-whales.

1 Problem Definition

Kaggle's Humpback Whale Identification Challenge asks participants to build an algorithm which can identify whales from images of their fluke. The provided dataset contains over 25,000 images of whale flukes, with 9850 of these images including labels and the remaining 15610 images serving as the competition's test set. Fundamentally, the task is a classification problem, where one is trying to design an algorithm which can classify an image into one of 4251 categories (with "new whale" being the class that is used when the algorithm believes it has not seen the whale before). The data and problem come from Happy Whale, an organization dedicated to helping people identify marine mammals from images [13].

Challenges with the dataset are several-fold. Difficulties include inconsistencies in image resolutions/sizes, images being in both color and black and white, flukes coming from many different perspectives, and many images including irrelevant contents (e.g., hand-writing). Of course, the largest difficulty to overcome is the lack of data for each class. Given that we have 4251 possible classifications but only 9850 labelled images, on average each class will only have 2 or 3 labelled instances. A notable exception is the "new whale" class, which has several hundred instances. Hence, there is also a data imbalance which our approach must account for. Given the lack of instances for most examples, an approach designed for few-shot learning ought to be used.

The following sections provide further details on the dataset and our methodology in solving the problem.

2 Dataset Exploration

As mentioned in the previous section, the training data consists of 9850 labelled (and 15610 test) whale images, where each label is one 4251 possible classifications. As shown in Figure 1, most classes have around 1 to 10 instances (with a few others having a couple dozen). The clear exception is the "new_whale"

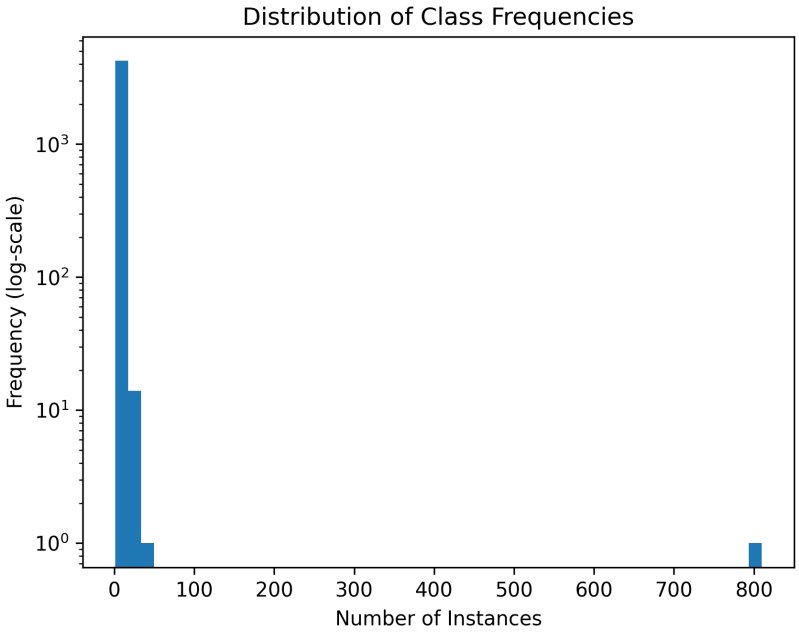

Figure 1: Distribution of classes in Happy Whale training data

class, which has 810 instances. The "new_whale" label does not apply to a particular species or whale, but rather is used to label images of whales that don't fall into one of the 4250 other classifications (e.g., a whale which has not been identified previously). Given the data shown in Figure 1, any good approach to this problem must account for the fact that there are not many examples for each class.

In terms of other characteristics, we found that the resolution of training images varied considerably. The minimum height and width of images was 30 and 64, respectively, whereas the maximum height and width were 1613 and 1050. Pixel values in each image are integers falling between 0 and 255. As well, the images have no clear consistency in terms of angle (of fluke), color, contrast, or background contents. Figure 2 shows sample images from the training dataset highlighting some of these characteristics. Given these inconsistencies, a good algorithm ought to be invariant to rotations, translation, contrast changes, and background details (i.e., any information aside from the fluke itself). As noted in [12], fluke features such as the notch, edges, and unique markings are most important for making a classification (see Figure 3), all of which are (in theory) unaffected by the inconsistencies described above.

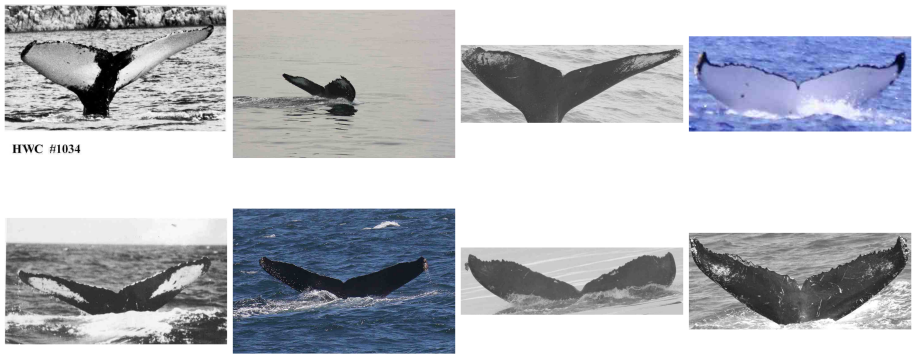

Figure 2: Sample images from training data

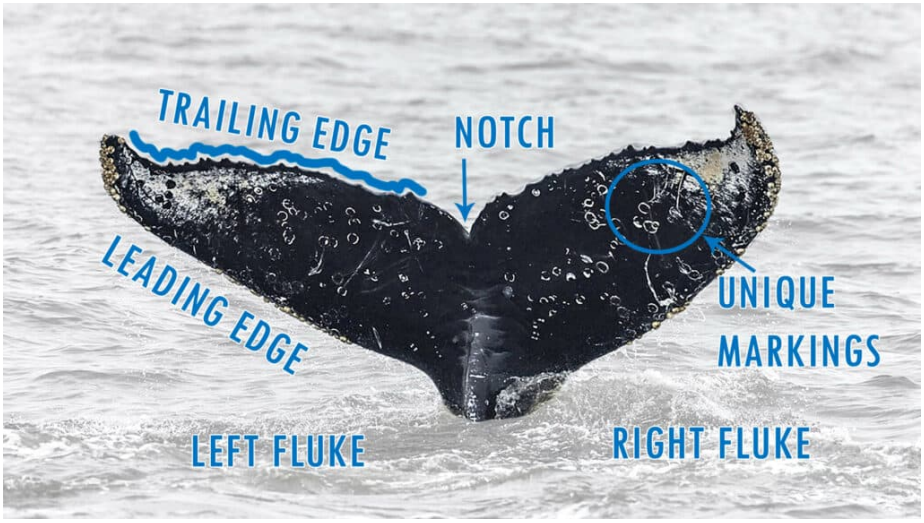

Figure 3: Most important features for identifying a whale from its fluke. Image taken from [12].

3 Data Preprocessing

Given the inconsistencies and challenges described in the previous sections, we deemed it necessary to normalize and resize images, as well as use various augmentations during training to improve model robustness. For all of these tasks we use the Albumentations library [3].

In terms of resizing, all images are resized to 256x256 using bilinear interpolation. Images are normalized according to the equation $img = (img - mean * max\_pixel\_value)/(std * max\_pixel\_value)$, where mean, std, and min/max pixel values are provided parameters [2]. In our case, we normalize using channel means and standard-deviations calculated from the 9850 training images (images are in RGB format and are normalized on a channel basis). Min and max values are 0 and 255, respectively. Given that some images are grayscale and others are in full color, we decided to standardize the coloring of images by converting all images to grayscale. We did not expect the conversion to grayscale to negatively impact model performance given that (in theory) this conversion should not impact important fluke-features described in the previous section.

Our data preprocessing also included several augmentation techniques. Given that we wanted our model to be invariant to rotation and flips, we use both affine transforms and horizontal flips during training to augment data. As well, in an attempt to improve the model's robustness to irrelevant image details we add gaussian noise, contrast adjustments, image bleaching, and other effects to images during training. It should be noted that each of these effects is added with some random probability, and thus we expect the model will be exposed to various transformations for each image over the course of training (e.g., affine transform + gaussian noise, image bleaching + contrast adjustment, no augmentation, etc). For further details on our augmentations please see the train_alb_transform() function in utils/transforms.py within our GitHub repository.

For training data we use all of the processing methods described above, whereas for validation and testing data we use only resizing, normalization, and making images grayscale.

4 Model Selection

A basic convolutional neural network was first adapted [9] to the whale data. It used the following structure: first, a convolution mapping from a grayscale to six channels over a 5x5 kernel. A second convolution of eight channels and a 5x5 kernel was used. To enhance feature selection, a 2x2 max pooling operation was performed after each convolution. Two fully connected layers were then added of size 550 (to avoid overfitting) and the number of output classes. ReLU activation was used on all layers except the last in order to provide nonlinearity.

It was learned that the whale dataset needed a model better suited for few shot learning since there aren't many samples per class. We came across the Siamese Neural Network with a sample implementation in PyTorch [10]. The

Siamese Neural Network aims to learn an embedding space for the images such that those with the same class have the smallest Euclidian distance. Therefore, Siamese Neural Networks require the following loss function (contrastive loss):

$$L = (1 - Y)\frac{1}{2}dist^2 + \frac{1}{2}Y(\max(0, m - dist))^2 \tag{1}$$

In Equation 1, Y is 1 if the output classes differ (0 otherwise), $dist$ is the Euclidian distance between two embeddings, and m is the max-margin as used in support vector machines. When the images are close in distance, but have different classes, the margin is enforced. In the training and evaluation sections below, Siamese Neural Networks performed well on whale classification even with the added margin hyperparameter.

In addition to the methods described above, we also tried to take advantage of Additive Angular Margin Loss (ArcFace) [5]. ArcFace is a loss function designed for facial recognition, a field where where few-shot learning is often necessary and models must be highly discriminative, and thus we believed it could be a suitable approach for this challenge. Rather than predicting the class of a given image or if two images come from the same class, ArcFace attempts to train the model to output similar embeddings for intra-class examples and dissimilar embeddings for inter-class examples. ArcFace loss is given by the following function:

$$L_{arcface} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cdot\cos\left(m+\theta_{y_i}\right)}}{e^{s\cdot\cos\left(m+\theta_{y_i}\right)} + \sum_{j\neq y_i} e^{s\cdot\cos\theta_j}} \tag{2}$$

Where m and s are the margin and scale (hyperparameters), N is the batch size, y_i is the label for class i, and θ_j is is the angle between the feature vector of the i-th sample and weight vector of the j-th class. The model used with ArcFace loss was an EfficientNet model pre-trained on ImageNet [11].

5 Model Training

The basic CNN model was trained using binary cross entropy loss on a 80/20 split of the training data. The Adam optimizer was used to take advantage of momentum and learning rate adjustments to combat residing in a suboptimal minimum [7]. Minibatches of size 16 were used to reduce memory usage. In addition, we use early stopping to cut model-training short if the validation loss does not improve after a given number of epochs (the "patience" hyperparameter). We store the model weights which achieve the lowest validation loss during train time and use these parameters for evaluation (see next section).

Where relevant the Siamese Network used a similar configuration as the basic CNN model. However, the training pipeline had to be modified in significant ways given the differences between a Siamese Network and a classifier such as the basic CNN. As described in the previous section, the Siamese Network learns by taking in two images and attempting to determine whether or not they

represent the same class, using the contrastive loss function for feedback 1. Since sampling image pairs completely at random would lead to an extreme imbalance (since most cases would be two images from different classes), we instead sample image pairs for the two cases (two images from the same class, two images from different classes) with 50-50 probability during training. Importantly, we do not allow for the "new whale" class to be used when sampling image pairs from the same class, as many examples in this class come from different whales.

Different backbones were used with Siamese contrastive training. This allowed for increased performance when using pretrained models. First, an Inception Resnet model pretrained on face data (VGGFace2) [6] was used as it was suspected that details were important rather than overall structure. A Xception model pretrained on ImageNet [4] was also used to investigate whether this expectation was valid.

Inception Resnet and Xception models have similar characteristics. Inception [14] aims to allow learning from locality regions within a CNN by using different sized filters and combining them with a new 1x1 convolution. Xception builds on top of Inception by allowing the new 1x1 filter to learn from filters on all of the input channels individually.

As with the Siamese Network, where relevant the ArcFace model used a similar configuration as the basic CNN. Little code change was necessary to convert the training of the basic CNN to an ArcFace model (changing the loss function and adding a few hyperparameters was all that was necessary), but it is important for one to keep in mind that the learning objective of the ArcFace model remains fundamentally different from the classification network. While the classifier is attempting to learn class distributions, the ArcFace model is instead attempting to optimize feature representations of input images to produce high-similarity embeddings for intra-class examples and distant embeddings for inter-class examples [5].

6 Model Evaluation

The basic CNN model utilized normal neural network testing. Each testing example was fed into the CNN, a partitioning of the top-5 indices was performed and saved to a Kaggle submission file. New whales, or those without specific classes, were treated as a separate class; thus, the ranked outputs solely determined what was predicted. Softmax was implicitly used in cross entropy loss, but, since order is preserved, partitioning occurred on logits. This likely proved to be ineffective compared to the Siamese approach below.

Given that both the Siamese Network and ArcFace model output embeddings rather than classifications, it was necessary to use a modified pipeline to generate predictions for these models. The basic idea is to generate embeddings for all training and testing images, and to generate predictions for test data by comparing the training and test embeddings. The prediction for a given test image will be the corresponding training label which gave the most similar embeddings. An important aspect as well is that we insert the prediction "new

whale" if the best similarity score is below a certain threshold, reflecting the fact that the model should predict "new whale" if it has low confidence that it recognizes the image. For the Siamese Network we generate pairwise distances between all train and test embeddings, choosing the top-5 predictions based on the labels with the lowest distance metric (and inserting "new whale" where the distance for the top-ith prediction goes above a certain threshold). For the ArcFace model we use the cosine similarity metric to generate similarity scores, choosing the top-5 predictions based on the labels which gave the highest similarity scores (and inserting "new whale" where the score falls below a certain threshold).

7 Hyperparameter Tuning

In an effort to prevent overfitting, for all models we use pre-trained weights from ImageNet and freeze layers except for the final classification layer. As well, we used L2 weight regularization and modified the value λ between 10^{-4} and 10^{-5}. Despite these efforts the ArcFace model and Siamese Network tended to still overfit. Figure 7 shows the training and validation for an ArcFace model using the regularization methods described, with the (smallest possible) EfficientNet B0 model. As can be seen from the figure, the model continues to learn from the training data while the validation loss plateus after around 5 epochs.

Since Inception and Xception were pretrained, they both exhibited rapid overfitting. Layers were frozen to maintain the convolutional filters from pretraining. Freezing all layers except the fully connected portions yielded the best results. Early stopping was also critical as 5-20 epochs seemed to be ideal for Xception, but Inception generated more correct top-5 classifications with 97 epochs (Xception performed better with fewer epochs). An embedding size of 500-1,000 (Inception/Xception) ensured that relevant features were extracted similar to PCA.

In addition, for the ArcFace model we found that adjusting the scale and margin hypeparameters used the loss function had a significant impact on model training. Our best results came from setting the scale parameter to $\sqrt{2}*\log(C-1)$, where C is the number of classes (C=4251), following the advice of [15]. Given the large number of classes we found setting the margin parameter to the small value of 10^{-5} had the best results. However, despite these adjustments the ArcFace model did not do much better than simply using an all "new_whale" submission at test-time, indicating that our model did not learn much during training (see 1).

Table 1 below shows the highest test scores achieved by each of the approaches described above. As a baseline, we submitted a prediction file containing only "new_whale".

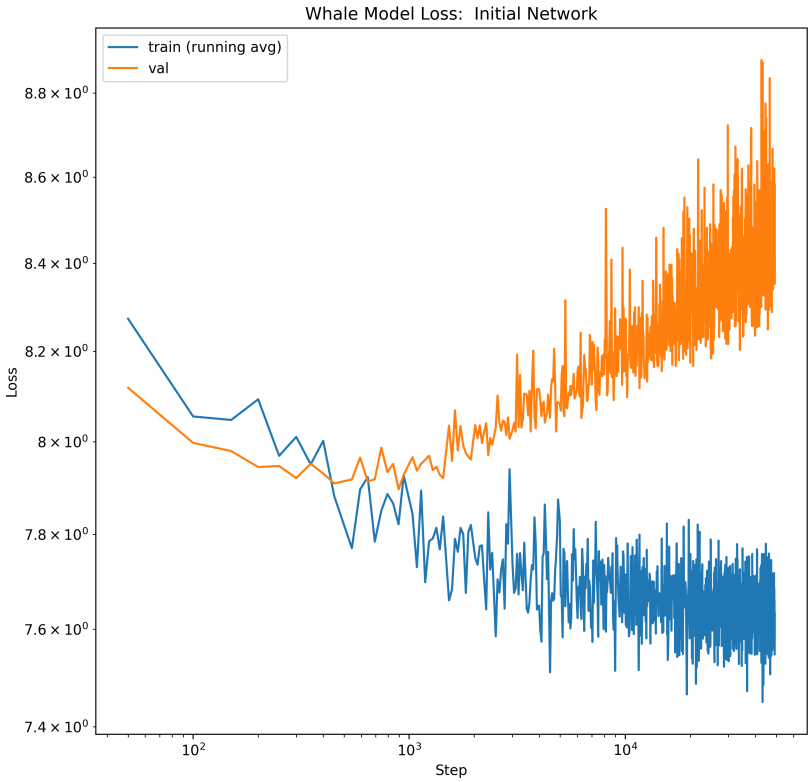

Figure 4: Train and Validation loss for basic CNN model over the course of training.

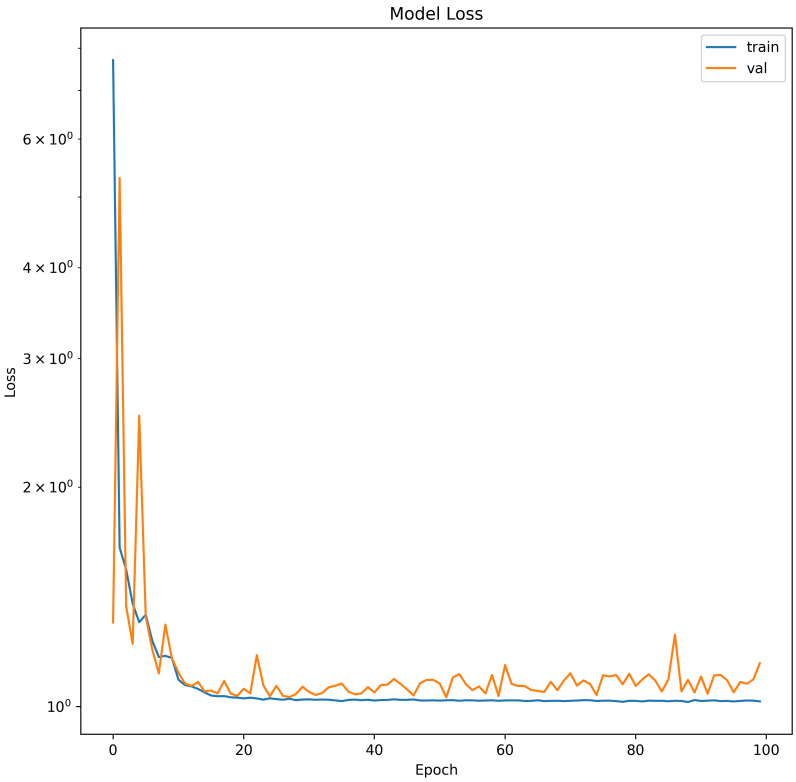

Figure 5: Train and Validation loss for Inception model pretrained on VGGFace2 data over the course of siamese training.

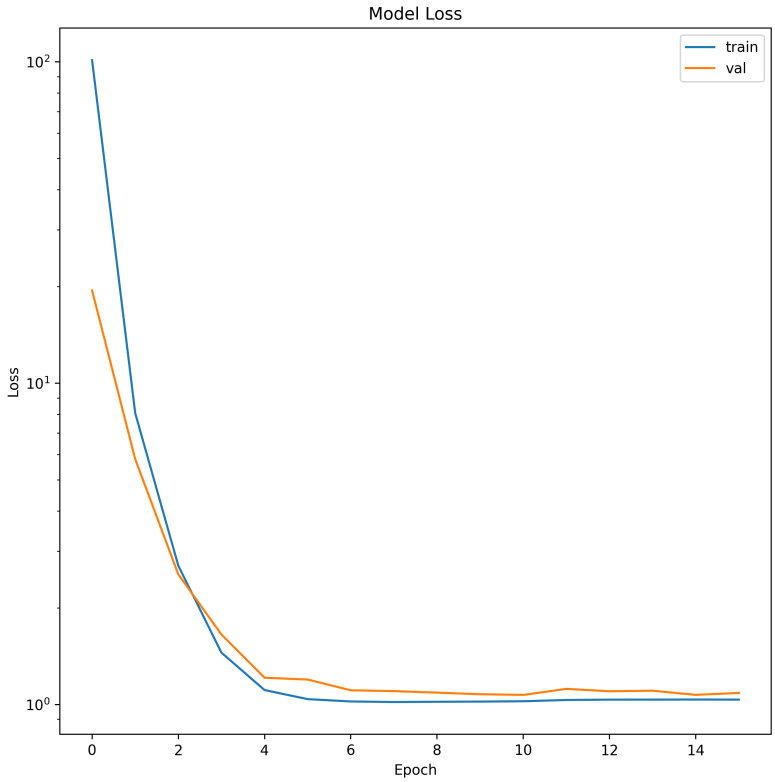

Figure 6: Train and Validation loss for Xception model pretrained on ImageNet data over the course of siamese training.

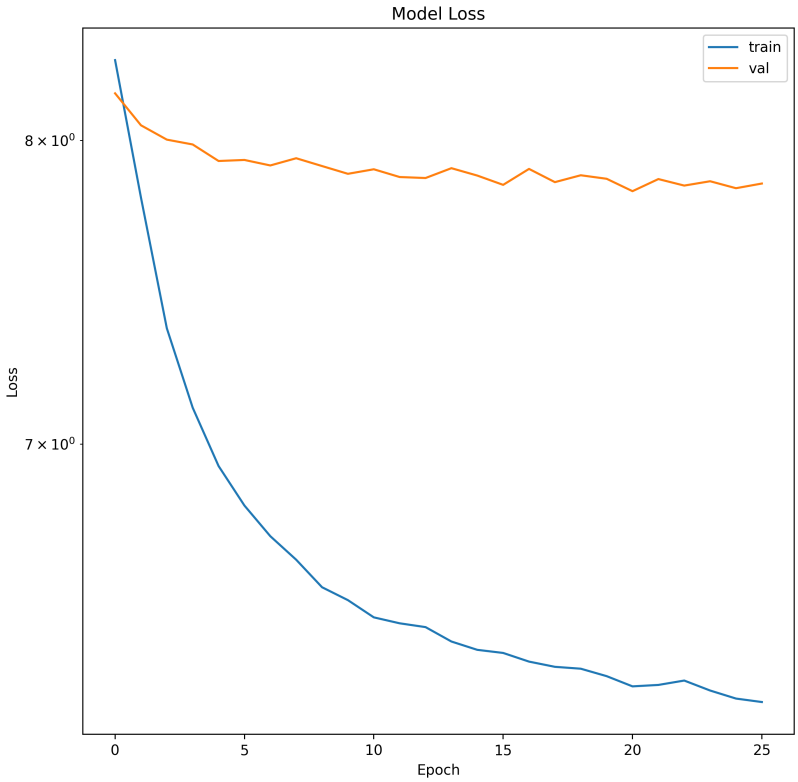

Figure 7: Train and Validation loss for ArcFace model over the course of training.

Approach	Highest Score
All "new_whale" (baseline)	0.32453
Siamese Net (Inception)	0.36497
Siamese Net (Xception)	0.37156
ArcFace	0.32979

Table 1: Overview of the highest test scores for different approaches

Figure 8: Kaggle submission with highest-score.

8 Conclusion

Overall, our best approach to the problem achieved a score of 0.37156 on the testing set, beating both benchmarks of 0.32786 and 0.36075. However, there are several areas where our work could be improved. We list some ideas for improvement below:

- Using a tool such as Optuna [1] to tune hyperparameters could have allowed for us to reach a more optimal set of hyperparameters
- Similarly, testing additional model architectures may have allowed for improved performance
- Using an object detection model to crop images around solely the whale fluke would have improved performance, as this method would help remove irrelevant details from many of the images
- Given the inherent difficulty of the problem, using a modified training pipeline which first feeds the model "easy" examples and then moves on to "hard" examples would help the model converge (especially the Siamese Network). In fact, the Kaggle user who received the highest score (0.78) on this competition several years ago noted that perhaps the most important aspect of their approach was to use such a training pipeline [8].
- Using an ensemble approach (i.e., averaging the predictions of several models) would likely provide more robust predictions.

The screenshot show in Figure 8 is of our highest-accuracy submission on Kaggle.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter

optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[2] Albumentations. Albumentations documentation - transforms (augmentations.transforms). https://albumentations.ai/docs/api_reference/augmentations/transforms/#albumentations.augmentations.transforms.Normalize.

[3] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020.

[4] Remi Cadene. Pretrained models for pytorch. https://github.com/Cadene/pretrained-models.pytorch, 2020.

[5] Jiankang Deng, Jia Guo, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. *CoRR*, abs/1801.07698, 2018.

[6] Tim Esler. Face recognition using pytorch. https://github.com/timesler/facenet-pytorch, 2023.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[8] Martin Piotte. Whale recognition model with score 0.78563. https://www.kaggle.com/code/martinpiotte/whale-recognition-model-with-score-0-78563, 2018.

[9] PyTorch. Training a classifier. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html.

[10] Strahinja Stefanovic. Siamese network in pytorch with application to face similarity. https://datahacker.rs/019-siamese-network-in-pytorch-with-application-to-face-similarity/#contrastive-loss-function.

[11] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[12] Amanda Townsel. Whale fluke identification: A visual guide. https://www.oceanicsociety.org/resources/whale-fluke-identification-guide/, Nov 2021.

[13] Happy Whale. Happywhale. https://happywhale.com/home, 2019.

[14] Joyce Xu. An intuitive guide to deep network architectures. https://towardsdatascience.com/an-intuitive-guide-to-deep-network-architectures-65fdc477db41, 2017.

[15] Xiao Zhang, Rui Zhao, Yu Qiao, Xiaogang Wang, and Hongsheng Li. Adacos: Adaptively scaling cosine logits for effectively learning deep face representations. *CoRR*, abs/1905.00292, 2019.