Humpback Whale Identification Challenge: An Overview of the Top Solutions

Henrique da Fonseca Simões   João Meidanis

April 29, 2021

Contents

Chapter 1

Introduction

The *Kaggle* platform hosts many competitions in the area of Machine Learning. In each competition, solutions are ranked by quality on public and private leaderboards [Kag19a].

Public leaderboards provide publicly visible submission scores based on a representative sample of the competition's data. Private leaderboards, on the other hand, are revealed only at the end of the competition along with the subset of data used to calculate them, and determine the competition winners. As a result, models that perform well on the public leaderboard, but not on the private leaderboard, are probably overfitting.

In the case of the Humpback Whale Identification Challenge [Kag19b], the public leaderboard contained 22% of pre-selected examples from the test dataset, while the private leaderboard had the other 78% of testing data points. For this competition, the evaluation function used to determine correct labeling was the Mean Average Precision at five (MAP@5), which is given by

$$\frac{1}{U}\sum_{u=1}^{U}\sum_{k=1}^{\min(n,5)} P(k)\cdot rel(k), \tag{1.1}$$

where U is the number of images, $P(k)$ is the precision at cutoff k, n is the number of predictions per image, and $rel(k)$ is an indicator function equaling one if the item at rank k is a relevant (correct) label, and zero otherwise. For this competition, a label is considered relevant just once for an observation u. This means the value scored for a correct prediction first appearing at the i-th position is equal to i^{-1}, with $i \in [1, \min(n,5)] \subset \mathbb{N}$. As a result, if all correct labels are among the top-2 predictions, one achieves a final score greater than or equal to 0.5.

Besides this competition, in 2018, one year before its launching, a *playground competition* on the same subject took place in *Kaggle* [Kag18]. Playground competitions are a type of "for fun" competitions that usually include small cash prizes, and are mainly targeted at newcomer competitors [Kag19a]. For the Humpback Whale Identification Challenge, the playground competition had its own training and testing datasets. This fact was leveraged by some solution developers during the later challenge.

The choice of participants' solutions analyzed here is based on their ranking on the private leaderboard, and on availability of solution code and training procedures,

Place	Team	Score
1st	Jian Qiao & Peiyuan Liao & Thomas Tilli & Yiheng Wang	0.97309
2nd	Tao Shen	0.97208
3rd	Jinmo Park	0.97113
4th	David Austin	0.96783
5th	Roman Solovyev & Weimin Wang	0.96781

Table 1.1. Top-5 ranking on the private leaderboard in the Humpback Whale Identification Challenge.

as presented in Section 1.1.

Since our goal is also to try to reproduce the selected solutions' trained models and their performances, we present in Section 1.2 the computational resources used in our experiments. The remaining chapters of this report are organized as follows. In Chapter 2 we present some statistics and peculiarities regarding the competition dataset. Chapters 3 through 5 detail each selected solution and our results trying to reproduce their network models.

1.1 Private Leaderboard and Candidate Solutions

At the end of the competition, the private leaderboard was published, and the top-5 ranking had the configuration shown in Table 1.1. In line with the Deep Learning community culture, most top ranking solutions were made publicly available by the competitors, some of which are objects of analysis in this report.

The first-placed team was one that shared their solution. This team was composed by four members, as shown in Table 1.1. Their real identities could be found through their public social media. Their solution is the first one to be analyzed, in Chapter 3. In Chapter 4, the second-placed solution, developed by Tao Shen, is analyzed. In Chapter 5, Jinmo Park's solution, which got the third place, is discussed.

1.2 Computational Resources

In order to try to reproduce the training results analyzed in this report, a server with an Intel Xeon Silver 4110 Octa Core 2.10GHz processor, with 126 GB RAM memory, and configured with 3 NVIDIA GeForce RTX 2080 Ti GPUs, with 11 GB of memory GDDR6 each, was used (see Section 6).

Most software packages needed to run the algorithms could be installed using the Docker Engine [Doc20] from community built images made available through DockerHub. In our experiments, we used the image *Deepo*, which has many frequently used Deep Learning packages installed [Yan17].

Chapter 2

Competition Dataset

This chapter offers an overview of the data used by competitors to train and test the algorithms created to solve the problem of identifying humpback whales by their tails [Kag19b]. The aspects considered in this analysis are the distribution of the examples over the classes (i.e., number of pictures for each individual whale), the image properties, and issues with non-standardized images, photograph angle, and occlusion.

The competition dataset contained thousands of images of humpback whale flukes. The data was divided in two parts: one for training the machine learning algorithm — with images manually labeled with an ID — and another for testing it — which did not include any labels. Competitors were required to deal only with the training data in order to develop their solutions, and use the test data only for submitting the results. See Table 2.1 for the exact number of images in each of those datasets.

All data information collected throughout this report was obtained from the *Kaggle* platform: the image files themselves and the training data labels.

2.1 Training Example Class Distribution

In order understand the example distribution among the classes, we used the labels available for the training data points, which contained the correspondence between image files and the class they belong to. The number of times a given whale ID appears in the labeling gives directly the number of examples available for that

Dataset	Number of examples	Percentage
Train	25361	76.11
Test	7960	23.89
Total	33321	100.00

Table 2.1. Number of examples per dataset and corresponding percentages relative to all data points.

Examples	Occurrences	Examples	Occurrences	Examples	Occurrences
1	2073	7	86	13	14
2	1285	8	76	14	16
3	568	9	62	15	19
4	273	10	46	16	16
5	172	11	39	17	17
6	136	12	26	18+	81

Table 2.2. Occurrences of the number of examples per whale ID available in the training set.

whale. With this information, we obtained the number of occurrences for each class size (see Table 2.2).

Even though there was a reasonable number of examples for training and testing, the training dataset did not have the same number of examples for each whale. Probably the same thing occurred with the testing dataset. Moreover, there were considerably more data points for some whales than others. Around 87.3% of the whales in the training dataset had less than six photographs, and 41.4% had only a single image for their identification (see Figure 2.1). Besides, around 38.1% of the images are pictures of unlabeled whales, collectively classified with the *new_whale* tag.

2.2 Image Properties

One important piece of information for building a neural network is the shape of the data the network will be processing. Some image processing net architectures, such as fully connected nets, require a fixed size image as input. Therefore, it is useful to analyze image size variation over the dataset, and other image properties, such as contrast, which can force the network to account for unimportant details, if no normalization is done. For this part of the analysis, train and test datasets were processed separately, in order to identify possible differences that could challenge competitors.

2.2.1 Image Sizes

Both training and testing datasets had a broad variety of image shapes, some of them more frequent than others. In the combined dataset, the most frequent image size was 1050×700 pixels. However, images with that size represent only around 13.77% of the data points available. Other image sizes that were also very frequent are shown in Table 2.3.

The remaining examples have dimensions distributed as shown in the Figure 2.2. Image width is less variable than height. Roughly 70% of the images are between 970 and 1150 pixels wide in both training and testing sets. On the other hand,

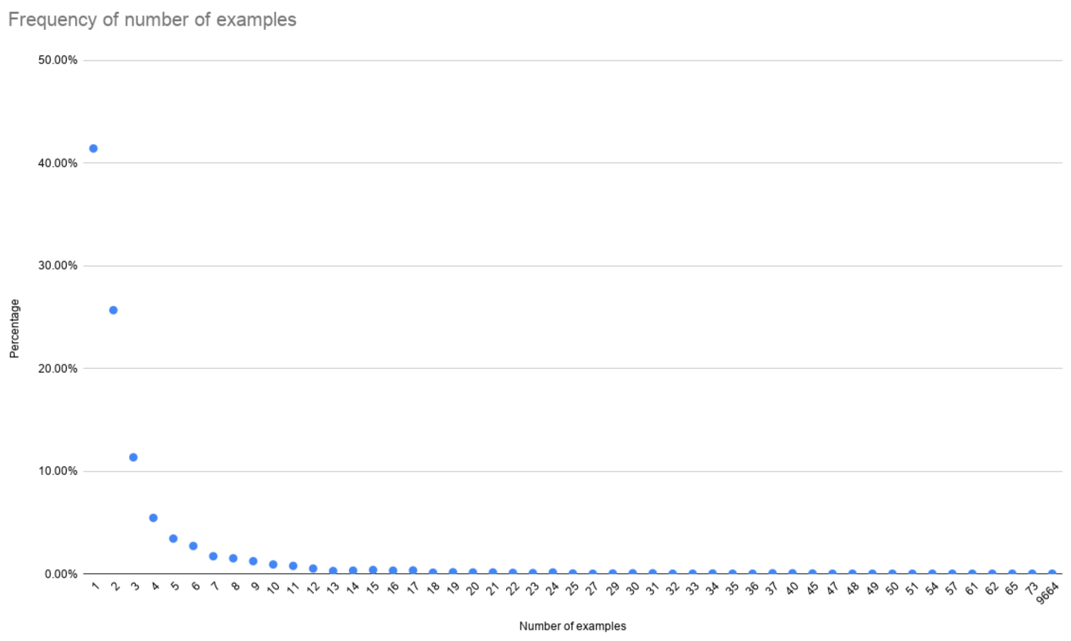


Figure 2.1. Frequency of number of examples per whale in the training dataset.

image height was more broadly distributed, with most heights ranging from 200 to 900 pixels. This image ratio makes sense, because the whale tail usually covered the image width almost entirely, while height varied, mainly due to cropping and tail angle before diving.

2.2.2 Image Color

Another relevant aspect for image classification when there are few examples per class is image color. Black and white images have the same information across the color channels. This means the network will not be able to use information about color hue to classify a grayscale image.

To detect image color, we used the *Pillow* Python library [LC19] to open the image and convert it to RGB format. If the image is already in this format, nothing is done. Otherwise, i.e., the image is in gray scale, all the RGB pixel values are set to the same intensity, preserving the original image content. With all images in RGB format, the image will be in gray scale if, and only if, every pixel has the same intensity across all its color channels.

In the analyzed dataset, most images were colored, but nearly 30% of the images were in gray scale (see Table 2.4).

2.2.3 Contrast

Contrast, in the context of Deep Learning, refers to the standard deviation of the pixels in an image or region of an image [GBC16]. More specifically, the contrast in

Width	Height	Percentage	
		Train	Test
700	500	2.63%	1.81%
1050	450	6.14%	6.58%
1050	525	5.14%	5.11%
1050	591	1.10%	0.98%
1050	600	10.05%	7.89%
1050	700	13.13%	15.80%
1050	701	1.04%	1.87%

Table 2.3. Image shapes in the train and test datasets with more than 1% of occurrence in any of them. Most images have a width of 1050 pixels, with varying heights. Some images are off by a single pixel in one of the dimensions, probably due to inaccurate cropping.

Dataset	Gray scale		Colored	
	Images	Percentage	Images	Percentage
Train	8270	32.61%	17091	67.39%
Test	2089	26.24%	5871	73.76%

Table 2.4. Number of images colored and in gray scale in the training and testing datasets and their corresponding percentages in the dataset.

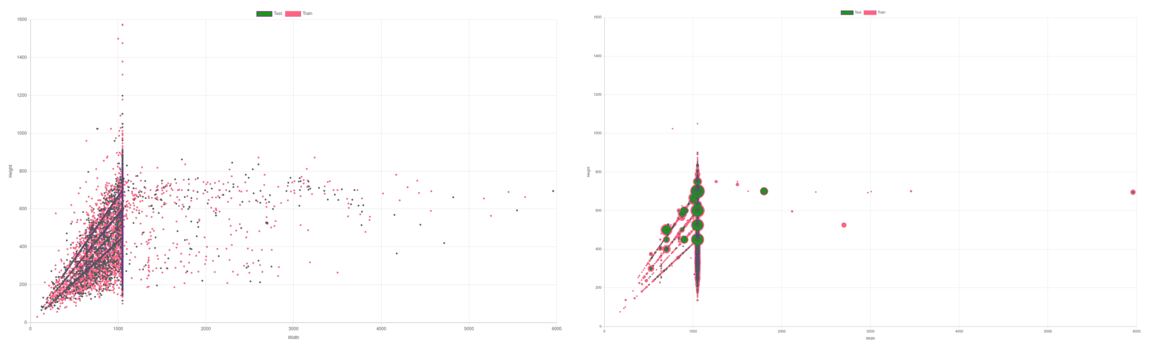
the entire image is given by

$$\sqrt{\frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3}\left(X_{i,j,k}-\bar{X}\right)^2}\,, \tag{2.1}$$

where X is the image represented as a tensor, with the first index representing the row, the second the column and the third the color channel in RGB and $\bar{X}$ is the mean intensity of the entire image:

$$\bar{X} = \frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3} X_{i,j,k}. \tag{2.2}$$

Although not directly related to the task at hand, contrast may still play an important role in the task solution, because the network may have to expend extra effort to conclude that contrast is irrelevant. When a high number of examples



(a) (b)

Figure 2.2. Image size distribution in the train (pink) and test (green) datasets. *(a)* Variety of sizes (regardless of frequency). *(b)* With the bubble radius proportional to the logarithm of the frequency. Note that a significant set of the data are 1050px pixels wide (vertical line on the charts); in addition, it is noticeable that three sets of images differ only in terms of the re-scaling factor (diagonal lines on both charts).

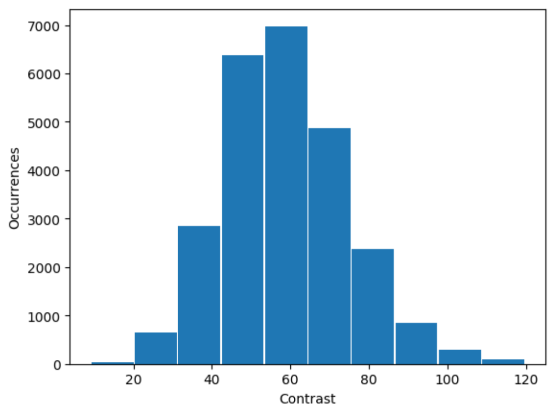
are available, this is often not a problem, since the model will probably learn which kinds of variability it should be invariant to. However, with only a moderate amount of examples available, as in this competition' dataset, dealing with contrast may be needed to eliminate avoidable generalization error.

In order to analyze contrast, we calculated it for each example available in each dataset using the *NumPy* Python library's built-in algorithm for figuring the standard deviation over tensors [Num19]. Since grayscale and colored images were present in both datasets, as shown in Section 2.2.2, all images were first converted to RGB color format, as described previously. Note this does not change the standard deviation value over grayscale images, since standard deviation across a single color dimension image (i.e. with 8-bit pixels values, black and white 'L' format) is given by

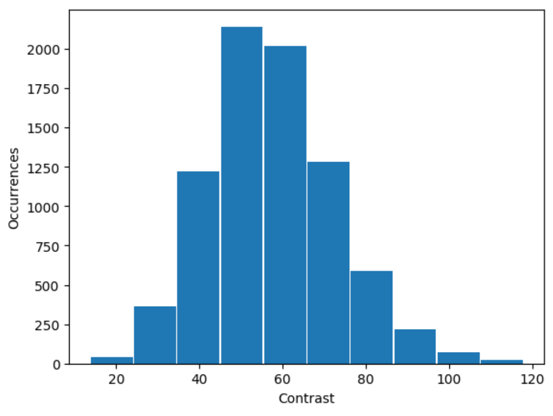
$$\sigma = \sqrt{\frac{1}{rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\left(X_{i,j}-\bar{X}\right)^2} \tag{2.3}$$

$$= \sqrt{\frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}3\cdot\left(X_{i,j}-\bar{X}\right)^2} \tag{2.4}$$

$$= \sqrt{\frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3}\left(\tilde{X}_{i,j,k}-\bar{X}\right)^2} \tag{2.5}$$



(a)



(b)

Figure 2.3. Contrast distribution of the images in (a) the train dataset, and (b) the test dataset.

where $\bar{X}$ is given by

$$\bar{X} = \frac{1}{rc}\sum_{i=1}^{r}\sum_{j=1}^{c} X_{i,j} \tag{2.6}$$

$$= \frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c} 3 \cdot X_{i,j} \tag{2.7}$$

$$= \frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3} \tilde{X}_{i,j,k} \tag{2.8}$$

and $\tilde{X}$ is the image tensor representing the image after the conversion to RGB, with the original pixel value replicated over the three RGB color channels, which is exactly the same as Equation 2.1.

The results obtained for image contrast are shown in Figure 2.3. Most images have a considerably high contrast value, with a mean value of 60 pixels of contrast. This shows that candidates who applied global or local contrast normalization might have achieved better generalization in their algorithms.

2.3 Bad Quality Images

Among all the available images, some were really unusual and could raise issues for participants to deal with. Malek Badreddine [Bad19], who was a participant in the competition, found some examples of problematic images in the training dataset. Some of these are shown in Figures 2.4, 2.5, 2.6, and 2.7, which are all classified as new whales. Another outstanding example is the one shown in Figure 2.8, which is actually a whale drawing with a specified ID. In this case, at least, there are ten other pictures of the same whale available in the training set, which probably helped dealing with it. Interestingly, one of these ten images is also problematic, and illustrates the issue of poor photo angle (see Figure 2.9). The remaining examples found by Badreddine are available in Appendix A.

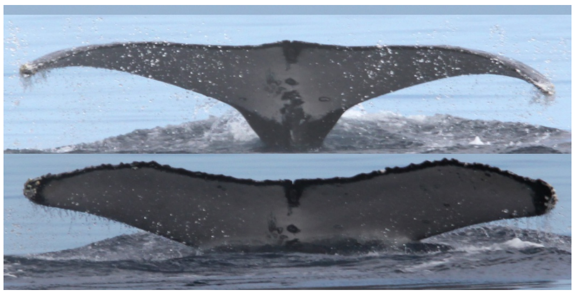


Figure 2.4. Two pictures, probably of the same whale, in the same image file.

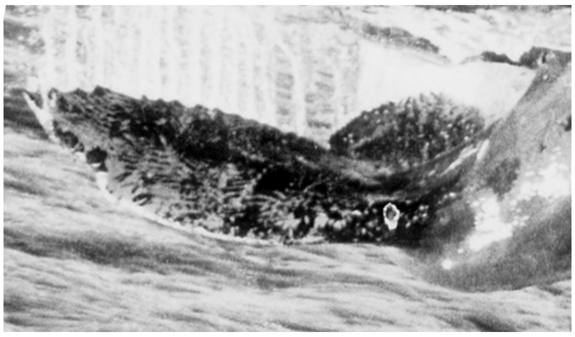


Figure 2.5. Picture of the front side markings of the fluke instead of the back side.

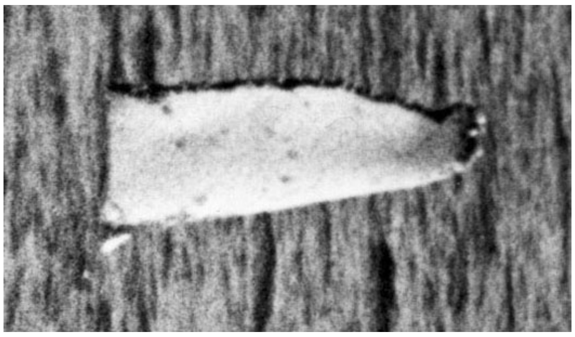


Figure 2.6. Noisy image and only half of the fluke exposed in the image.

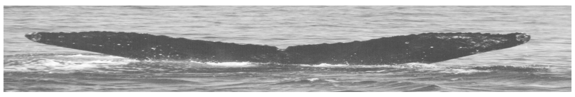


Figure 2.7. Poor aspect ratio ($2528 \times 382\,px$).

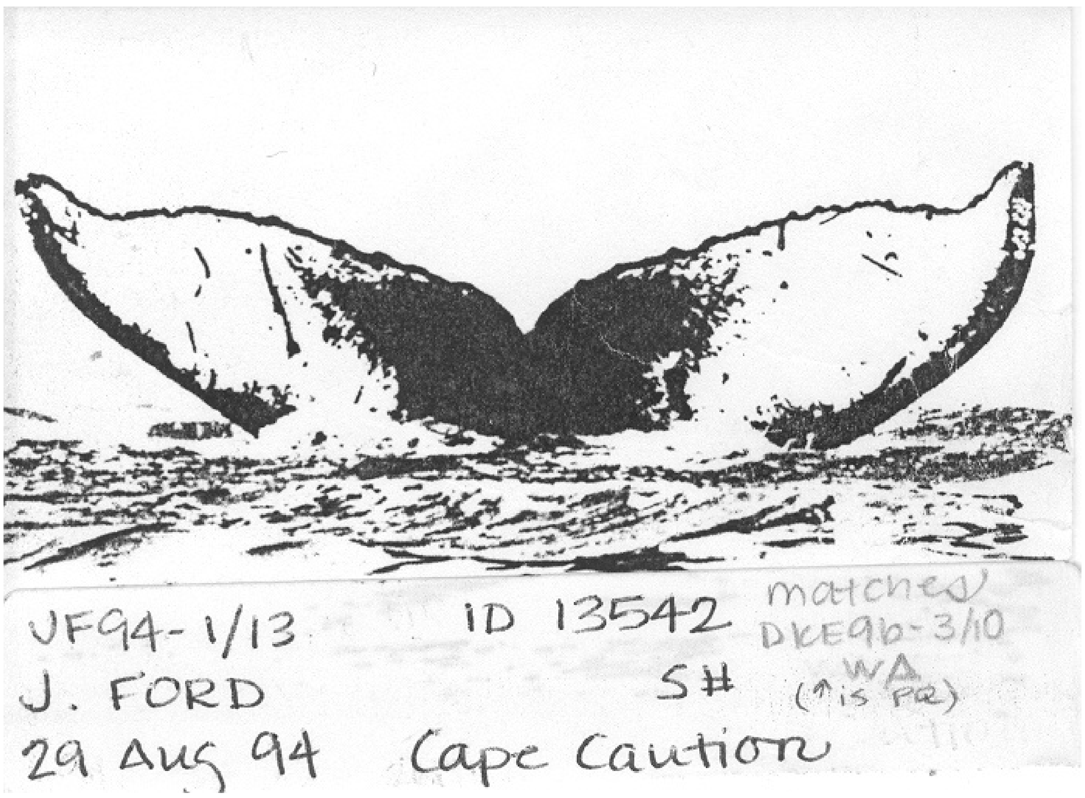


Figure 2.8. Whale sketch with handwriting.

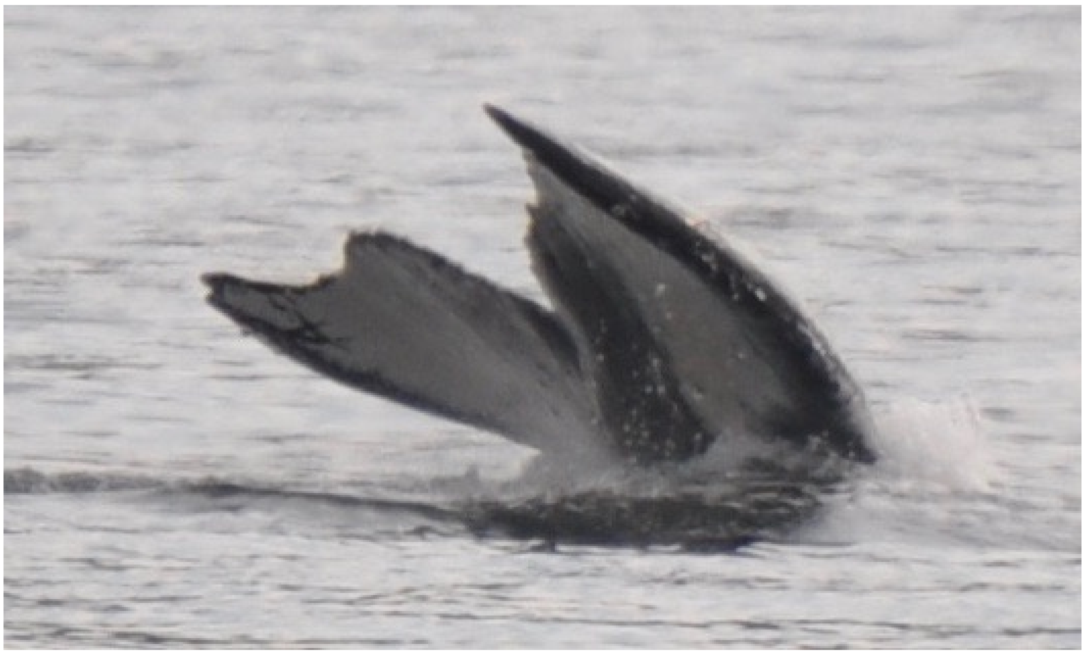


Figure 2.9. Problematic photo angle for the same whale that was sketched.

Chapter 3

First Place Solution

Strategies and a brief explanation of the network used in this solution were published on the *Kaggle* Forum [Qia19], and authors' code was made available through the *GitHub* platform [QLTW19].

As described in the Qiao *et al.* solution post, they used many different techniques to solve the problem, some of them aimed at increasing a few percentage points in the score. In next sections we analyze the solution from different aspects, including data treatment, network architecture, and strategies used to increase generalization.

3.1 Data Treatment

The strategies used regarding the competition data were essentially:

- building masks identifying the whales' tail in the images;
- using detection to crop the images to be processed;
- standardizing image sizes; and
- augmenting the data using several different transformations.

The next sections discuss each strategy more closely.

3.1.1 Image Masks

In order to help the network identify important parts of the images, *i.e.* the flukes, Qiao *et al.* created masks for each image available in the dataset (see Figure 3.1). These were used as a 4^{th} input channel with the images' usual RGB channels being the other three.

The neural network probably used to create the masks, according to an answer to another competitor in the solution discussion forum [Qia19], was the U-net. This network is based on the idea of performing several convolutions and pooling, reducing the image dimensions while doubling the feature channels in each pooling stage, and then performing an expansive path, consisting of an up-sampling of the feature map followed by an "up-convolution", and a concatenation with the corresponding cropped feature map from the contraction stage [RFB15].

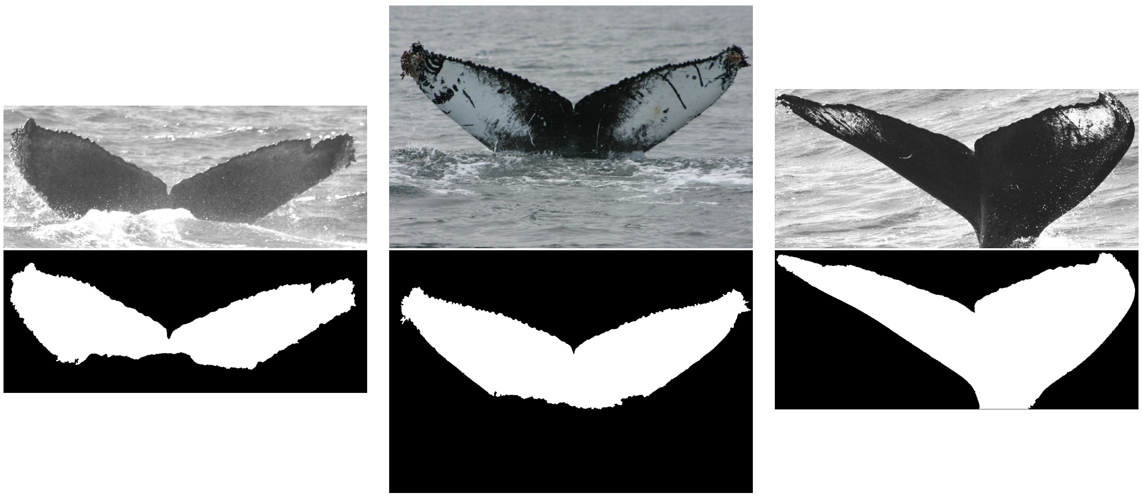


Figure 3.1. Some masks created to facilitate the process of identifying a whale by its tail.

3.1.2 Bounding Boxes

Another strategy used was the creation of bounding boxes to remove useless background information. The exact method used to obtain these boxes is not detailed by the winners, but a basic processing on the masks should be enough to define the boxes, since the masks have all the informative pixels set as white, and all others set as black.

3.1.3 Standardized Image Sizes

As shown in Chapter 2, one concern about the data was image sizes, which varied considerably. The solution developed by Qiao *et al.* dealt with this problem by standardizing the dimensions during image load. The input tensor used in the final version had dimensions (4, 512, 256), where the first entry represents RGB + mask channels, and the last two entries corresponds to width and height, respectively.

It is important to state that the resizing operation was done always after applying the bounding boxes cut. This way, the entire whale's fluke fit the image used as input to the network, even though the image was originally distorted and not centered.

3.1.4 Data Augmentation

A standard method to increase the network performance on classification tasks is regularization through data augmentation, which is particularly effective for object recognition [GBC16]. This strategy consists in creating new fake data using the ones available in the dataset by applying operations like translation, rotation, and scaling.

In the case of the solution provided by Qiao *et al.*, several operations were applied to images with different probabilities (see Table 3.1). However, unlike usual practice, these new images were not stored in the hard drive as new files. Instead, they were

Transformation		Probability
Random erase		0.5
Random shift		0.5
Random scale		0.5
Random angle rotation		0.5
Mask deletion		0.5
Noise	Speckle	0.25
	Gaussian	0.25
Brightness shift		0.125
CLAHE		0.125

Table 3.1. Image transformation applied to images for augmentation, and their probability of being applied to each image during training.

created and stored on volatile memory, used by the learning procedure, and then discarded.

Some operations applied to images had no effect, due to parameter values that make the operation an identity function. Those have not been considered in this analysis as data augmentation.

Next, a more detailed explanation of each transformation in Table 3.1 is given, showing the parameters and the effects they produce.

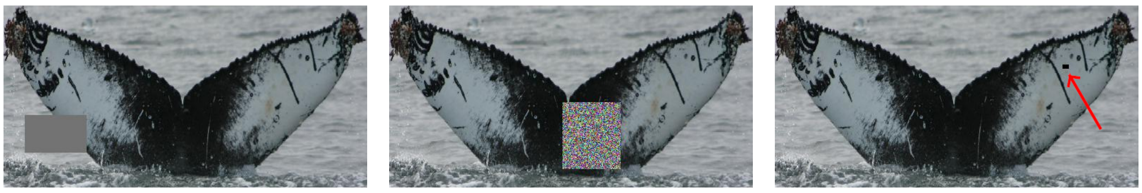
Random Erase, Shift and Scale

Random erase is a data augmentation technique intended mainly to solving the issue of occlusion, increasing the network robustness [ZZK+17]. The erasing procedure consists in randomly choosing a rectangular part of the image and resetting its pixels. One may use random or mean image values to replace the original pixels, or even zero the pixels. Some examples of the resulting images are shown in Figure 3.2.

The strategy used by Qiao *et al.* was selecting a random position in the image being processed as the start position of the erasing, and a size for the rectangle to set the values as zeroes. The rectangle size was randomly chosen within the range $[5, 10]$, resulting in an erasure of at most 0.076% of the image information.

Another approach, related to erasing, is shifting. It consists of changing each pixels' position (i, j) to a position $(i + \epsilon, j + \delta)$ for some small values $\epsilon, \delta \in \mathbb{Z}$. The randomly chosen values for ϵ and δ used by Qiao *et al.* lie within the ranges $[-10, 10]$ and $[-15, 15]$, respectively. The part of the image left uncovered by the move had its pixel values set to zero.

Even though it leads to part of the image becoming invisible, similarly to erasing, the main purpose of this method is to make the network invariant to translation. This means the first layers of the net will be capable of identifying the presence of



(a) Mean value method (b) Random filling method (c) Zero filling method

Figure 3.2. Examples of the random erase applied to images from the Humpback Whale Identification Challenge. In the first two examples, a rectangle with dimensions in the interval $[50, 100]$ was used to make the visualization easier. In the third example, the method used by the winners was applied, in order to illustrate the effect of the transformation. The erasing (black square) is on the upper right fluke side (arrow) in the third example. All images were first cut using the bounding boxes created for this whale photograph and then resized to $(512, 256)$.

features that characterize a certain whale even if the photograph is not taken exactly at the same position in which the algorithm was trained.

Yet another method used was random (re)scale, consisting of randomly choosing a new scale for the image and resizing it. Similarly to the other approaches, they used zero-filling outside of the image's placement, which was also randomly chosen. As scaling factor, a value was randomly chosen between 90% and 100%, which implies that most of the image was still the whale's tail.

Since masks were also being used, all these transformations were applied exactly the same way to the masks, to keep consistency.

Random Rotation

Image rotation has a similar goal as image translation, but is probably more important in terms of data augmentation. This is because Convolutional Neural Networks are naturally invariant to small translations, due to their convolutional kernels being applied one next to the other and therefore sharing a great deal of information about the image region. Effects similar to rotation might be accomplished by convolutions in deeper layers of the network, but require the network to learn these transformations [GBC16]. In this sense, the winners decided to apply the transformation directly to the image, letting the rotation angle be randomly chosen within the range $[-25, 25]$ in degrees.

Mask Deletion

Even though masks were created for all images, the winners decided to sometimes let the network figure out on its own how to extract information about the whale's flukes in the image without using the masks. This was accomplished by literally erasing the mask layer by setting all its values to zero.

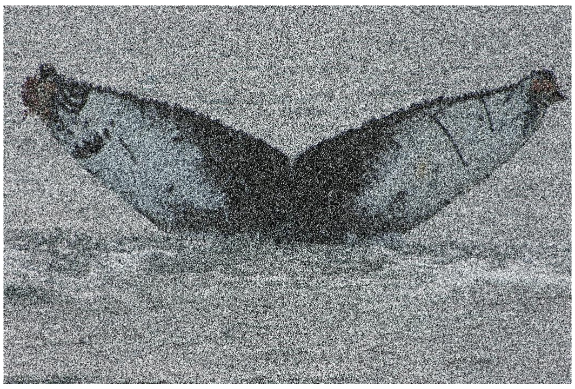


Figure 3.3. Example of an image with Gaussian noise injection with $\sigma = 0.5$ and $\mu = 0$.

Noise Injection

Noise is defined as unwanted signal that disturbs the original signal due to physical phenomena linked to the acquisition and transmission of the information. In the context of Image Processing and Deep Learning, different mathematical functions might be used to artificially produce noise for using as data augmentation or even as a regularization strategy [KC17]. Here we discuss Gaussian noise, originally used to model thermal noise effect on electronic devices, and speckle noise.

Mathematically, Gaussian Noise is a type of additive noise given by the formula

$$\hat{s}(x) = s(x) + \eta, \tag{3.1}$$

where $\hat{s}(x)$ is the resulting signal, $s(x)$ is the pure signal, and η is a random variable drawn from a Gaussian distribution given by

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \tag{3.2}$$

where σ and μ are called noise parameters. Usually μ, which represents the mean of the distribution, is set to zero, and the noise is controlled only by the standard deviation σ. In the case of an entire image, the values of η are usually drawn independently for each pixel in the image.

This is essentially the approach used by Qiao *et al.*, configuring $\sigma = 0.5$, and adding noise to the luminosity channel of the image transformed from RGB to CIE-LAB [CF97]. This means that around 68% of the noise added to the image will be within the range $[-\sigma, \sigma] = [-0.5, 0.5]$, in a context where the maximum and minimum pixel value lie in the interval $[0, 1]$.

In fact, this might be seen as a strong noise being added to the image, as shown in Figure 3.3, and could lead to a network failing completely to recognize the image if it is not trained to deal with it [KC17].

As shown by Koziarski and Cyganek [KC17], noise injection as a form of data augmentation can significantly increase the generalization of the Deep Learning algorithm if the type of noise which the network will face is present in the dataset. Otherwise, in the case where the image signals are virtually pure, the performance might even be worse if the injection is done during training.

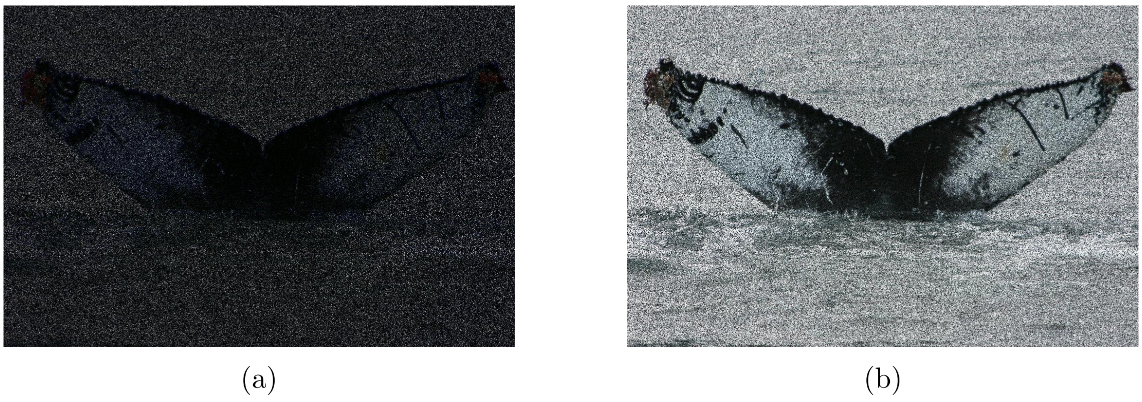


Figure 3.4. Difference between noisy image using (a) the standard modeling of speckle noise, and (b) the modeling used by the winners.

However, considering the different types of image quality composing the Humpback Whale Identification Challenge dataset, such a downside effect would not be expected.

Differently from Gaussian noise, speckle noise [AB07] is modeled as a multiplicative noise. In other words, it is given by

$$\hat{s}(x) = \eta \cdot s(x), \tag{3.3}$$

where $\hat{s}(x)$, $s(x)$ and η are respectively the resulting signal, noise-free signal, and noise value drawn from a normal distribution (Equation 3.2) with zero mean ($\mu = 0$) and variance one ($\sigma = 1$).

In the case of the speckle noise used by Qiao *et al.*, the multiplicative term was slightly different, given by

$$\eta = 1 + \tilde{\eta}, \tag{3.4}$$

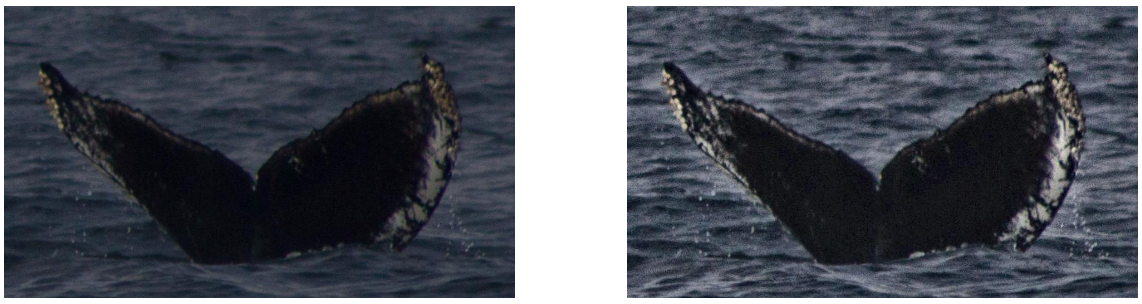
where $\tilde{\eta}$ was drawn from the normal distribution independently for each pixel value in the image. Using this approach, most pixel values do not get as dark as they would with the transformation given by Equation 3.3. A comparison of both approaches is shown in Figure 3.4.

Brightness Shift

The transformation named brightness shift by Qiao *et al.* was the addition of a constant value to all pixel values in the image, *i.e.*, it was given by

$$T_{i,j,k} = X_{i,j,k} + \alpha \tag{3.5}$$

where T and X are tensors representing respectively the transformed and the original images, with the first two dimensions representing the spatial dimensions, and the third the color channel. In the final submission code, the constant value α was set to 0.1, an increase of 10% on the maximum possible pixel value.



(a) Original image (b) CLAHE

Figure 3.5. Example of enhancement provided by applying CLAHE.

Contrast Limited Adaptive Histogram Equalization (CLAHE)

Adaptive Histogram Equalization (AHE) is a technique used in Image Processing to correct image contrast by applying to each pixel in the image the histogram equalization mapping based on a contextual region. This method has produced great results in medical imaging, and was further developed to enhance even more the quality of the images. One important enhancement done was the introduction of the clipped or contrast-limited AHE (CLAHE), which limits the level of contrast enhancement to prevent overenhancement of noise [PAP+86].

Basically, the idea is to redistribute the image pixel values to a more uniform distribution, limiting the amount of pixels that can have the same value. This means that an image with nearly uniform regions, and thus high peaks in the histogram, can be greatly improved (see Figure 3.5).

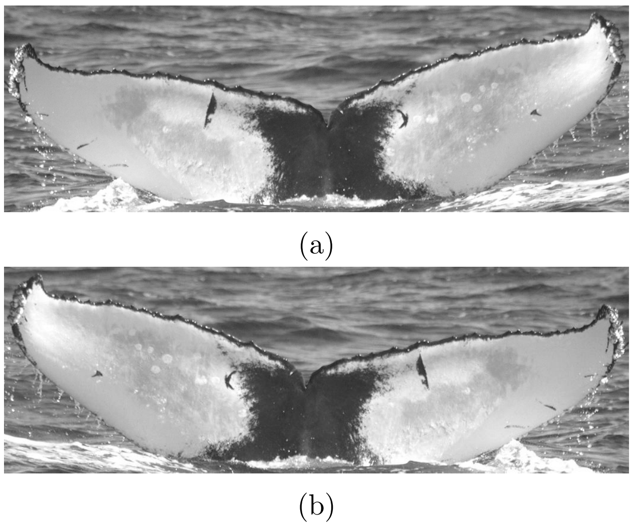
3.2 Special Techniques

Some (unexpected) techniques were used in the first place solution to increase some few score points on the classification result. Among them, we cite introducing new images to the dataset, creating fake new whales, and even application of post-processing based on class prediction frequency.

3.2.1 Flipped Images

Even though horizontally flipped images do not keep the label, since most whales' tail marks are not symmetrical (see Figure 3.6 for an example), Qiao *et al.* found it useful to augment the data with such a transformation. The idea came from a competitor post on the Kaggle Forum [Che19].

Following this idea, all images were flipped during training and inference phase, having a new class assigned to them if not classified as new whale. Flipped new whales remain new whales. For inference, flipped image prediction was used to balance the original image prediction, by averaging both of them to generate the final classification. In this averaging process, only the new classes were considered for the flipped image, while only the original classes were considered for the original


(a)

(b)

Figure 3.6. Example of non-symmetrical marks on whale's tail. (a) Original image. (b) Horizontally flipped image.

image. Moreover, the vectors were added in a way that the original entry for a given whale matched its counterpart's entry.

3.2.2 Pseudo Labels

This strategy consisted in using a trained model with high confidence on its predictions to assign labels to training data from the playground competition. In this process, Qiao *et al.* augmented the training dataset by 1774 images with labeling confidence greater than 0.96.

3.2.3 Class Balance

After some investigation on the whale classification, Qiao *et al.* observed a correlation between number of labels and scores. Thus, they introduced a post-processing phase in their classification to further balance their predictions.

After the output was generated by their network, the classification produced for images was analyzed, and the following procedure was applied. For a given image, if the absolute network confidence difference between predictions for the first and second class was less than 0.3, and the second class was not used in any top-1 prediction, and the first class was used in top-2 predictions "many times", first and second class were switched.

In other words, if the network was in doubt about which class to assign to a given image, they kept the classification when either the second class was always chosen to be the top-1 prediction for other images, or the first class was not used many times as the top-2 prediction.

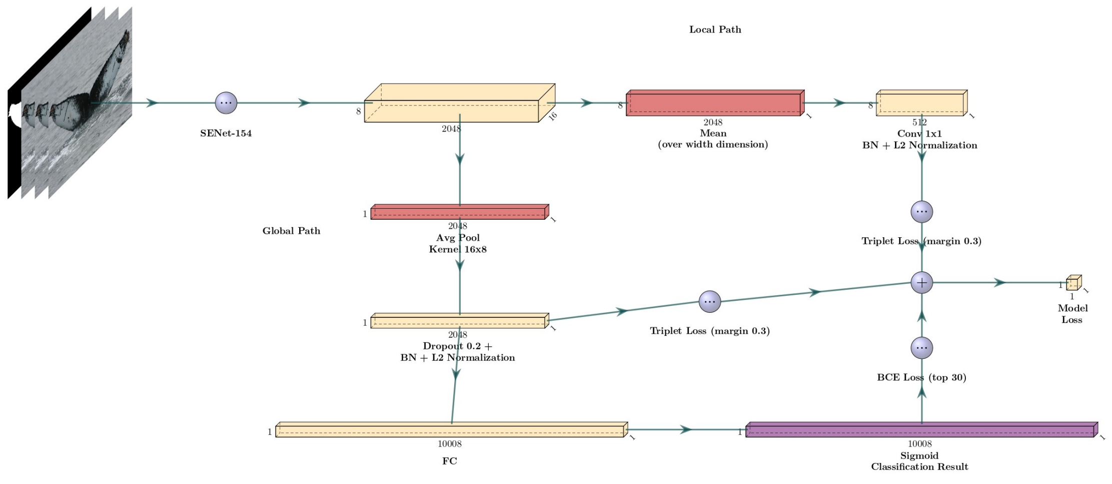


Figure 3.7. Overall neural network architecture used by Qiao *et al.*

3.3 Network Architecture

During the competition period, several network architectures were tested by the winners, including simpler solutions such as softmax classifiers with fixed threshold. However, some of them did not perform as well as expected, requiring them to find different approaches. In the end of the competition, the final architecture used by them was composed of a backbone SENet-154 [HSS18], with local and global feature extraction as the last network layers.

The network input was standardized to be of size $(4, 512, 256)$ as discussed in Section 3.1.3, and this was directly fed into the backbone network as shown in Figure 3.7. This network has, as shown next, 2048 feature maps of size $(16, 8)$ each as output, in this case. Using this output, global and local features could be extracted to generate the network loss and the classification output. Further details on each of these steps are given in Sections 3.3.2 and 3.3.3.

3.3.1 Backbone Network

The base network used in the final submission was a Squeeze-and-Excitation Network with 154 layers (SENet-154). This network was proposed by Jie Hu *et al.* as a modification of the 64×4d ResNeXt-152 [XGD+17] — which adds block stacking to a residual network, namely ResNet-152 [HZRS16] — by incorporating a unit called "*Squeeze-and-Excitation*" (SE) block into it [HSS18].

More details on the SENet-154 architecture design with the input and parameters used by Qiao *et al.* are given next. To simplify the explanation, first we describe how the building blocks work, and then how more complicated structures are built using these blocks, raising the level of abstraction.

SE Block

This special network building block is capable of modeling interdependencies between channels. Basically, the strategy used here is to capture global information

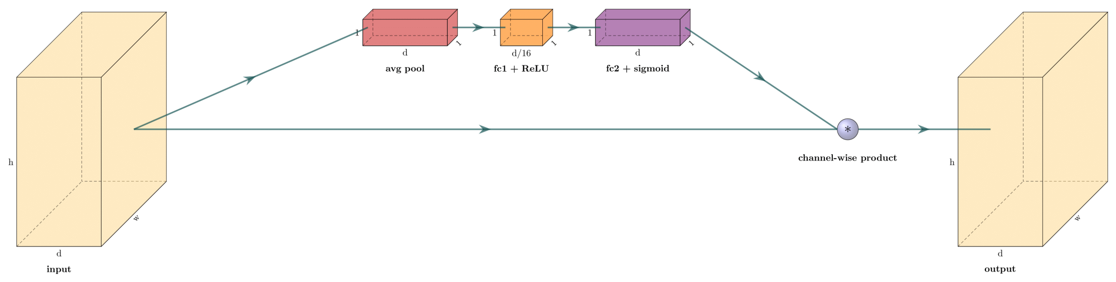


Figure 3.8. SE-Block operations.

about each input channel — in this case, by average pooling — and to pass this information through fully connected layers with some kind of non-linearity such as ReLU. After that, a vector intended to indicate the relevance of each channel is obtained and used to weight the original block's input. Applying this strategy was shown to increase performance of the entire network [HSS18]. See Figure 3.8 for an architectural sketch.

SE Bottleneck

Before passing features through the SE Block, a few convolutions are done, possibly changing the spatial dimensions of features, on a block called SE Bottleneck. This series of operations is used several times on the network with different values for the stride s and the number of output planes $4d$, with $s, d \in \mathbb{N}$.

As apparent in Figure 3.9, the number of output planes is always $4d$, but sometimes the input does not match this number of features. Since this part of the network uses a residual learning framework, the residual, *i.e.* the bottleneck input, has to be added later to the SE Block output. In order to make this possible, a downsampling convolution is done, making the dimensions once again compatible. The same happens when the spatial dimensions are changed by a convolution with stride greater than one. This residual step used in these paths is mainly responsible for preventing such a deep network from degrading the training accuracy [HZRS16].

Overall SENet-154 Structure

The SENet-154 network uses the principle of stacking multiple blocks to build several layers. In this case, the entire net consists of five main layers, each one occupying a row in Figure 3.10.

The first one performs certain operations upon the raw input, in this case the image itself. These operations are convolutions that compute several features and reduce the spatial dimensions of the input, and pooling, as can be seen with further details in Figure 3.10.

This first step generates 128 feature maps, which are fed to the second layer. This layer performs three consecutive full rounds through the bottleneck path, increasing the number of features. Similarly, the next (third) layer uses eight stacked SE Bottleneck paths, having the feature dimensions reduced, while the number of features is increased. The next two layers use a similar strategy, further reducing the spatial

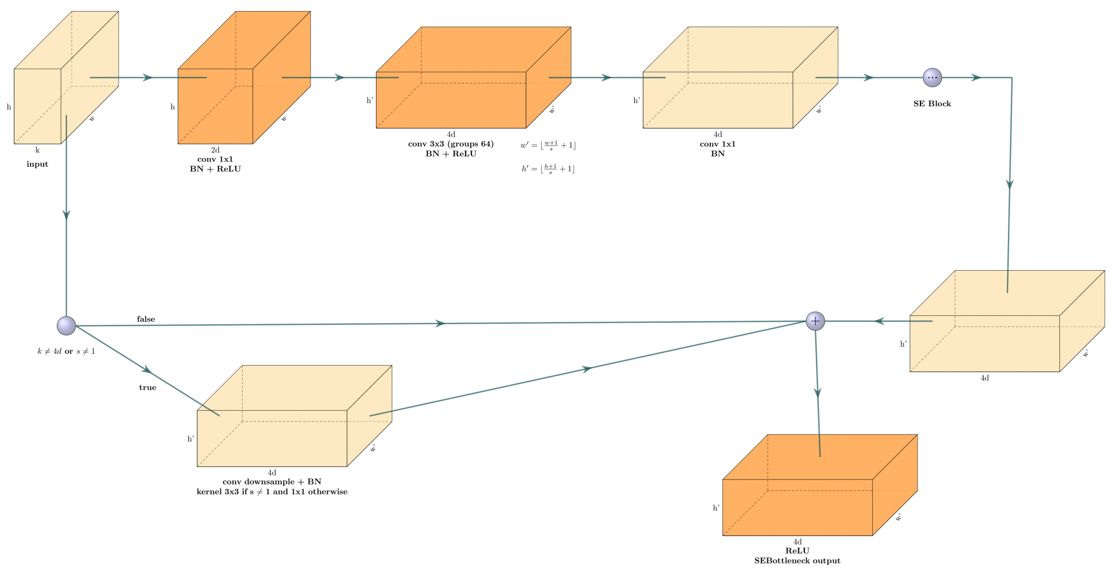


Figure 3.9. SE-Bottleneck model.

dimensions and increasing the number of feature maps, by using, respectively, 36 and 3 bottleneck paths.

The full process generates an output consisting of 2048 feature maps of size $(16, 8)$. In the original SENet-154, more steps would take place after this result, having a fully connected layer using dropout applied to it. However, these steps have been included in the final processing used by Qiao *et al.* as a global feature extraction, which is described next.

3.3.2 Local and Global Features

Among the custom parts of the network, local and global feature extractions are some of the most important ones. They compose the last layers of the network and are basically designed to generate the classification output using the features created by the backbone network, and calculate the loss associated with the entire model.

Local feature extraction is a component of the Part-based Convolutional Baseline (PCB) initially proposed in the field of person re-identification. It was created to conduct uniform partition on convolutional layers for learning part-level features, which are then processed to reinforce within-part consistency through the refined part pooling (RPP) [SZY$^+$18].

For whale identification, Qiao *et al.* used the PCB idea of uniform partition applied to the backbone network result to create a pipeline for the model loss and classification. The within-part consistency was not a concern for the solution implemented. Besides that, the number of horizontal stripes used was set to 8 instead of 6 (the recommended value). According to them, the choice of using horizontal stripes came from the hypothesis that this would make it easier to distinguish flipped images from their counterparts.

On the other hand, global feature extraction is essentially an average pool over the whole spatial dimension of feature maps from the backbone network, as shown

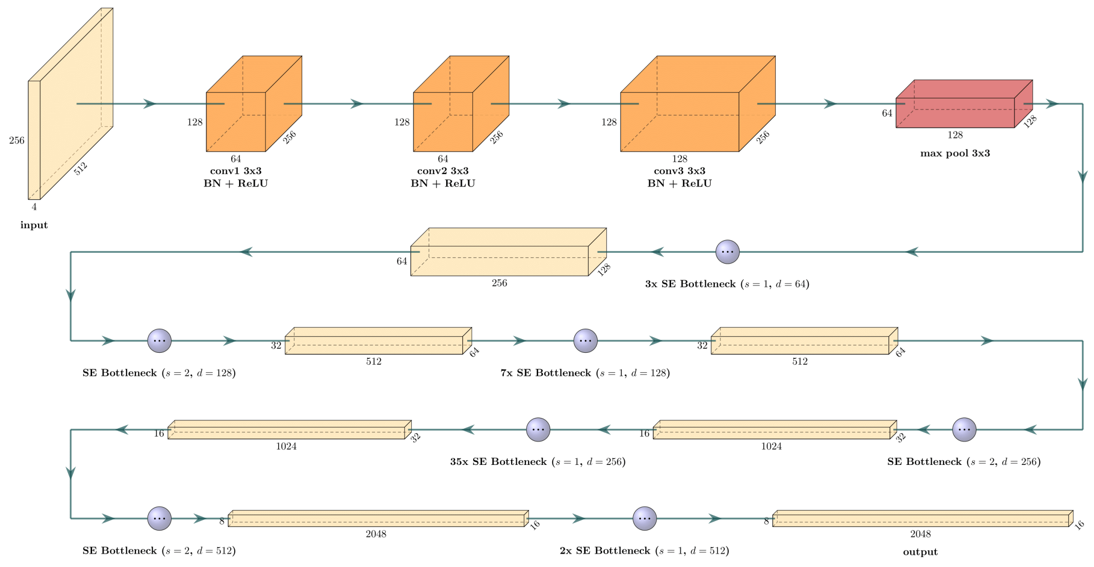

256
512
4
input
128
64
256
conv1 3x3
BN + ReLU
128
64
256
conv2 3x3
BN + ReLU
128
128
256
conv3 3x3
BN + ReLU
64
128
128
max pool 3x3
64
256
128
3x SE Bottleneck ($s = 1$, $d = 64$)
SE Bottleneck ($s = 2$, $d = 128$)
32
512
64
7x SE Bottleneck ($s = 1$, $d = 128$)
32
512
64
16
1024
32
35x SE Bottleneck ($s = 1$, $d = 256$)
16
1024
32
SE Bottleneck ($s = 2$, $d = 256$)
SE Bottleneck ($s = 2$, $d = 512$)
8
2048
16
2x SE Bottleneck ($s = 1$, $d = 512$)
8
2048
16
output

Figure 3.10. SENet-154 overall model. For each SE Bottleneck path used in the network, we included the parameters used in the step for the stride (s) and number of output feature maps ($4d$).

in Figure 3.7. These features, which can be understood as a summary of the local feature information, are directly used in a fully connected layer without bias to generate the classification vector, which then passes through a sigmoid activation and is used to predict which classes are more likely to be assigned to the whale in question.

This way, global features might be seen as the usual final layers from a conventional SENet-154, which include a global average pool, followed by dropout with dropping rate of 0.2, and a fully connected layer. Yet the softmax classifier is not used by the competitors, who opted for a sigmoid function applied to the last layer output instead. Another difference relies on the fact that the classification vector is also used to compute the loss, along with global and local features. This is described in Section 3.3.3.

3.3.3 Loss Functions

Two different loss functions are used to generate the model loss: Triplet Loss and Binary Cross-Entropy (BCE) Loss. The first is used upon the local and global features, while the second upon the classification result. Then, both are added, resulting in the final loss term. In the sequel, each term is described more precisely.

(Hard) Triplet Loss

As discussed by Hermans *et al.* [HBL17], Triplet Loss, proposed by FaceNet [SKP15], came from a modification of the Large Margin Nearest Neighbor (LMNN) loss function created by Weinberger and Saul [WS09]. The latter is based on the idea of having the algorithm learn a linear transformation that roughly satisfies two criteria: points which belong to same class have to be close to one another and points from different classes have to be far apart by a margin. To accomplish this, “target neighbors” — points which belong to the same class — and “impostors” — points from other classes — are selected prior to the training for each data point, and those respectively contribute to a *pulling* and *pushing* term of the loss function. These terms lead the network to know and learn whether the clustering is good enough. Based on the transformation learned, it is then possible to apply the k-Nearest Neighbors (kNN) algorithm [CH67] to classify unseen data points.

Differently from the aforementioned, Triplet Loss does not require points from the same class y_i to collapse into a single point, but only to be closer to *positive* examples than *negative* ones, by at least a margin. Mathematically, this is achieved by describing the loss function as

$$\mathcal{L}_{tri}(\theta) = \sum_{\substack{a,p,n \\ y_a = y_p \neq y_n}} \max\left(0, m + D_{a,p} - D_{a,n}\right), \tag{3.6}$$

where θ is the parameter of the neural network function, a, p and n are the *anchor*, *positive* and *negative* examples respectively, m is the margin constant, and $D_{i,j}$ is the distance between i and j data points.

Hermans *et al.* show that using the (non-squared) Euclidean distance as metric works better than its squared version, due to stability. Moreover, they point out other issues regarding triplet selection for large datasets. For Triplet Loss to lead points of the same class into the same cluster, long enough training is necessary. When the dataset is too large, such training is infeasible. Thus, they presented a version of Triplet Loss which samples triplets only within the randomly sampled batch, and figures the loss using the hardest positive and negative examples among them. By *hard* positive and negative examples they mean respectively examples from the same class but with very little similarity and similar-looking examples from different classes. See Figure 3.11 for examples from the Humpback Identification Challenge.

It is important to guarantee the existence of triplets with *hard* examples, since showing easy examples repeatedly to the network will not be of great help after a long enough training time. Using the randomly sampled batch helps dealing with this, after all it will contain on average moderate examples, which might be seen as the best ones to use in general.

Qiao *et al.* followed this strategy, but instead of creating a batch using P classes with K examples each, as proposed by Hermans *et al.*, they used a slightly different approach. Every time a class was randomly chosen from the dataset, an anchor and a positive example were randomly loaded, along with an identified negative example and another without identification (*i.e.* from the *new whale* class). As a result, each batch had at least one positive and two negative examples. One might notice that, in this case, hard negative examples are more prevalent within the batch than hard positive examples, since the number of whales that share negative examples is greater within a batch sampled this way.

As shown in Figure 3.7, Triplet Loss is calculated over the local and global features. In both cases, hardest examples are found using the Euclidean distance and the loss is computed as in Equation 3.6 with margin $m = 0.3$.

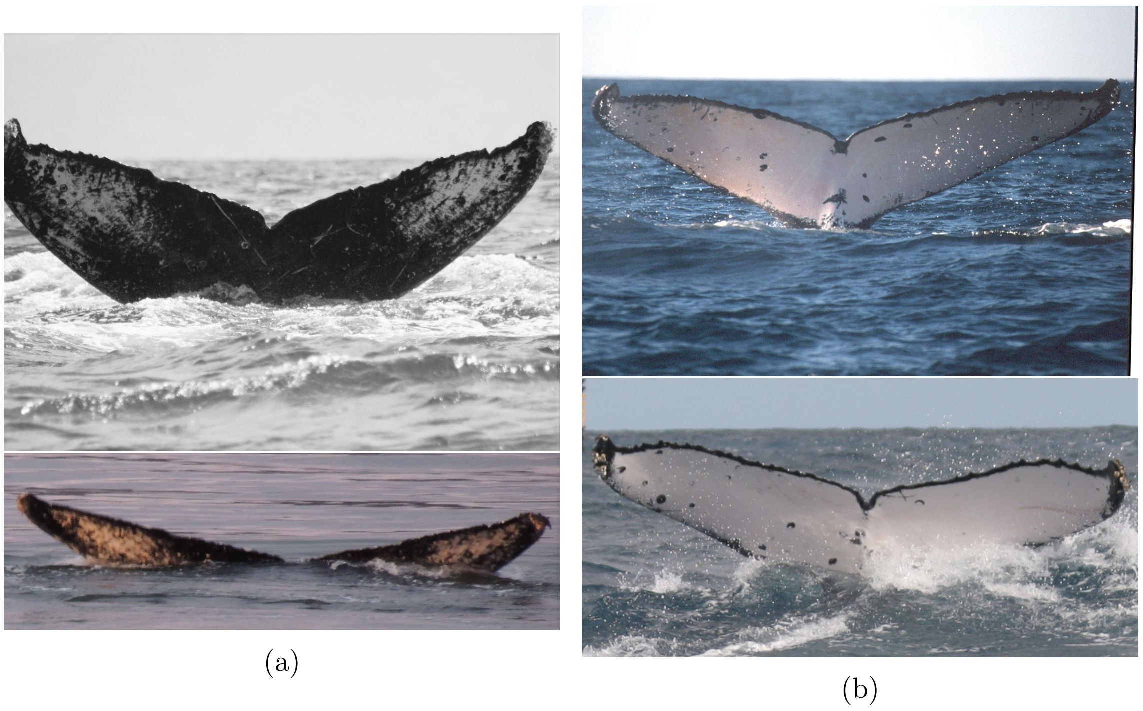


Figure 3.11. Hard examples part of the Humpback Whale Identification Challenge dataset. (a) Hard positive example: the image below belongs to the testing set, while the above is part of the training set. (b) Hard negative example: the above image, labeled as *w_c20b4e0*, belongs to the training set, while the test image (below) is from the *w_fd57399* class, which has a single training image available.

Binary Cross-Entropy Loss

Binary Cross-Entropy Loss is computed over the classification result after the sigmoid function has been applied to it, to guarantee that each entry in the classification vector will lie in the interval $[0, 1]$. As a consequence, the output values can be fed to the following function

$$\mathcal{L}_{BCE}(X, Y; \theta) = \frac{1}{Cn} \sum_{k=1}^{C} \sum_{i=1}^{n} -\left[Y_{k,i} \cdot \log X_{k,i} + (1 - Y_{k,i}) \cdot \log(1 - X_{k,i})\right] \tag{3.7}$$

where X and Y are the input (*e.g.* classification after being fed to the sigmoid function) and target tensors, respectively, with the first entry corresponding to the C batch examples, and the second corresponding to each of the n classes, and θ being the network parameter. In their solution, this loss was usually used twice upon the classification result, each time using different input and target tensors.

First, for each example in the batch, the absolute difference between the result of the sigmoid of the network classification output and the one-hot classification vector was calculated. Then, the top-30 highest error values for each example were placed in a tensor X, and fed to the BCE Loss function (Equation 3.7) with the target tensor Y with all entries zeroed. This produced a scalar that composed the error, which penalized high values for wrong classes and low values for the correct one.

Second, when there was at least one non-new-whale example in the batch, another loss term was calculated using only these examples and added to the final loss. This time, the input tensor X was defined as the network output after being fed to the sigmoid function, and the target tensor Y had all its entries set to 1. This means that the network was penalized once again if it wrongly did not attribute a high value for the identifiable whale classes.

Both these terms were then added to generate the final term for BCE Loss shown in Figure 3.7.

3.3.4 Training Procedure

Having in mind the highly imbalanced classes in this competition, the first-solution authors got inspiration for their training strategy on academic research dealing with this issue, along with other, standard strategies.

First, regarding the backbone model, one training strategy used was the transfer learning technique [PY09], which helps to decrease the training time and increase generalization. This approach is based essentially on the idea of importing parameters from a trained model which performs a similar task. Intuitively, this is believed to increase performance and decrease training time, since a pre-trained neural network is already able to identify features from the images that enable class separation. Thus, the network does not need to learn everything from scratch.

In case of the transfer learning used by the competitors, the model had its parameters configured from a SENet-154 network pre-trained on the ImageNet dataset [RDS+15, Cad17]: a large image dataset organized according to the WordNet hierarchy [Mil95]; in other words, a dataset that contains several images illustrating several nouns, including objects, animals, and geological formations.

Second, a similar strategy to the one proposed by Sun *et al.* [SZY+18] was also used to deal with local feature extraction training. This can be mainly seen from the fact that the model was trained using the following steps, which are mostly the same as the ones proposed by Sun *et al.*, except for the absence of a p-category part classifier.

- First, the model is trained using only classes that contain at least 10 images, to convergence. The convergence criterion was to achieve 0.98 on mean precision at five on the training set;
- Second, all but the two last layers of the backbone network are fixed (frozen) — feature extractors stay unfrozen —, and the model continues from the best place it has stopped, now using all the images available in the training dataset;
- Finally, a fine-tuning step is done from the best checkpoint achieved on the second step, still with the network layers fixed.

Besides that, inspiration from other papers can be found in this training procedure. In particular, the strategy of using only the images which contain at least some fixed number of examples can be found on a paper by Yandong Guo and Lei Zhang [GZ17], where they present certain techniques to be used on face recognition, when few training examples are available. These are called one-shot or underrepresented classes. Even though the whale identification challenge also faced underrepresented classes, most new ideas presented in Guo and Zhang's work are not included in the final solution developed by Qiao *et al.* However, as stated before, the strategy of training first on a *base set* — a set consisting only of classes with high number of examples —, and then on the set containing the one-shot classes is indeed used on the first-place solution.

Parameter Initialization

Most parameters used on the network were configured using transfer learning, as described previously, but the ones used on the last layers (global and local feature extraction) had to be initialized heuristically.

Fully Connected Layer. They had the fully connected weight matrix initialized with values drawn from a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 10^{-3}$, and biases all set to zero.

Normalization. Batch normalization bias was set not to be learned on either global or local path.

Convolution. Parameters in the convolution performed on the local path were set using the default initialization implemented by the PyTorch library, which samples for both weights and biases from the uniform distribution $U(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{C_{in} \cdot k_1 \cdot k_2}$, with C_{in}, k_1 and k_2 being respectively the number of input channels, first kernel dimension and second kernel dimension.

Hyperparameters

In terms of hyperparameters, the configuration used in the solution was the following.

Optimization Algorithm Parameters. The final version of the solution used the Adam optimization algorithm [KB14], with exponential decay rates for moment estimates $\beta_1 = 0.9$ and $\beta_2 = 0.99$, and weight decay coefficient $\alpha = 2 \cdot 10^{-4}$.

Learning Rate. This hyperparameter might be seen as one of the most important parameters to be configured correctly. As discussed by Goodfellow *et al.* [GBC16], overshooting the learning rate can lead to complete failure of the learning algorithm, as well as undershooting it, which leads to very little progress being made at each iteration — possibly making the algorithm stuck in a high training error point. In this case, the authors of the solution set this hyperparameter with value $\epsilon = 3 \cdot 10^{-4}$ during the first and second steps. On the last training phase, the intention was to fine-tune the optimization only, so a smaller learning rate was configured, $\epsilon = 3 \cdot 10^{-5}$.

3.4 Reproducing Solution Training

As shown earlier, the source code of the solution developed by the first-place competitors was made publicly available through the GitHub Platform [QLTW19], and contained all auxiliary files and scripts necessary to run the training procedures specified by the authors. However, this does not mean the exact same result can be reached, since learning processes are inherently stochastic.

Along with the files available in the GitHub repository, the authors used several images from the playground competition. These images had to be included in the original training dataset in order to have all labeled images in training folds available for training. More details about the folding is given in Section 3.4.1.

Next, we describe the results obtained with the trained models.

3.4.1 Training Steps

For getting as close as we could to the result achieved by Qiao *et al.* during the competition, we set to strictly followed the steps provided by them.

In this respect, we used the source code provided, altering only the batch size used by the training algorithm. This had to be done due to differences in memory and number of GPUs used by them versus what was available to us. As commented in a response to another competitor, they used 5 NVIDIA Titan X Pascal GPUs, with 12GB of memory GDDR5X each. This allowed them to use batch size 12, while our configuration permitted just 5. It is important to note that these batch sizes are not exactly the number of images processed by the network. For each batch example, three other images are loaded along with it: one positive example (another image of the same whale), and two negative examples (one from another identified whale and the other from the *new whale* class).

Recall that the training consisted of three steps:

- Convergence using examples with at least 10 classes with the network fully adaptive;
- Inclusion of all examples with all but last two layers frozen; and
- Fine-tuning with smaller learning rate.

All these steps were followed altering only the necessary parameters. This means the default (first) training fold was used. These folds were created by splitting the training data into two different sets: validation and training. Each fold contained all images originally available for training, differing among them in which images were present in each set. For all splits, the number of validation examples was 742, representing only around 3% of the data available for training. Another key difference among the folds was the presence or not of the playground images. All but the first fold had all labeled images from the playground competition training set, which consisted of 1774 images.

Hence, the default configuration provided by Qiao *et al.* did not include new training images at all. At first, we followed this approach, even though we knew that the authors used playground images in their solution. Then, we conducted another full training procedure, this time using the second fold — which did include the playground images. All the partial and final results for both fully training attempts are presented in the next section.

Model	Step	MAP@5		Top@1		Training time
		Train	Validation	Train	Validation	
First	1	0.995	0.3477	99.1	34.6	9h
	2	0.462	0.8323	35.7	76.4	14h
	3	0.623	0.8643	46.0	80.0	11min
Second	1	0.992	0.3747	98.6	37.3	10h
	2	0.467	0.8130	45.4	74.4	14h
	3	0.569	0.8724	39.9	81.3	18min

Table 3.2. Partial results on each training step of the trained models. *(Above)* First training model, using only original data. *(Below)* Second trained model, using extra labeled images.

Model	Leaderboard score	
	Public	Private
Qiao *et al.*'s	0.97461	0.97309
First	0.82299	0.85070
Second	0.84870	0.86513

Table 3.3. Official results on public and private leaderboards on Kaggle Platform for Qiao *et al.*'s final submission and our training attempts.

3.4.2 Results

Both trained models had their phases disjoint, in other words, checkpoints have not been shared between the model using playground images and the one did not. Moreover, all training steps were followed as described in Section 3.3.4 in both cases.

Partial results for each step for the first and second model are shown in Table 3.2.

The resulting model from the third step on both attempts was used to generate a classification for submission on the Kaggle Platform. The goal was to evaluate the trained model exactly the same way the competitors had evaluated theirs. An important aspect to note is that the post-processing step (class balance) was not applied before submitting the classifications to the platform, since neither its precise description nor its code had been given by the authors.

The classification score on the public and private leaderboards is shown in Table 3.3.

3.5 Conclusion

In this solution, several different strategies were used to overcome the challenges proposed by the competition, including heavy data augmentation using many transformations, creating masks, and using face recognition and person re-identification strategies. In general, the solution can be seen as metric learning and classification, differently from other high score solutions. In order to check the reproducibility of the model training, two different training procedures were performed: one not using pseudo-labeled data and the other using them. In both attempts, the scores were around 10 points below the ones achieved by the solution authors.

Chapter 4

Second Place Solution

Similarly to Qiao *et al.*'s, this solution's code was published on the *GitHub* platform [She19b]. However, a brief description of the solution and partial results obtained during the competition was created by Tao Shen in the code repository, not on the *Kaggle* Forum. In order to give further details about the solution — probably to the competition hosts and the community — and share thoughts and experiences applied to solve the challenge, the author made presentation slides available along with the code.

As shown on the presentation slides, Shen's prior-knowledge in the fields of face recognition and person re-identification based on Deep Learning helped him deal with some of the competition's challenges.

To present this solution, we use a similar approach to that used in Chapter 3. This means we first analyze Shen's approaches regarding the competition data, in Section 4.1. Then, a discussion on problem-specific techniques used in the solution ensues, in Section 4.2. In Section 4.3, we cover the network design, including the backbone networks, the loss functions, and the training procedure. Unlike the first-placed solution, Shen used model assembling as the last part of the classification pipeline, to further increase the prediction confidence. The ensemble model is described in Section 4.4. Finally, Section 4.5 shows our results using the solution algorithm.

4.1 Data Treatment

Data treatment usually involves augmenting the dataset, and choosing how to manipulate the data points. Shen decided to use two different image sizes — a specific feature of his solution. Besides, special care is taken to frame the important part of the image, *i.e.* the fluke, by creating bounding boxes. Moreover, many transformations are applied to the images to augment the data and help the network account for important variation factors, probably improving generalization.

More information about these tactics is presented in the next sections.

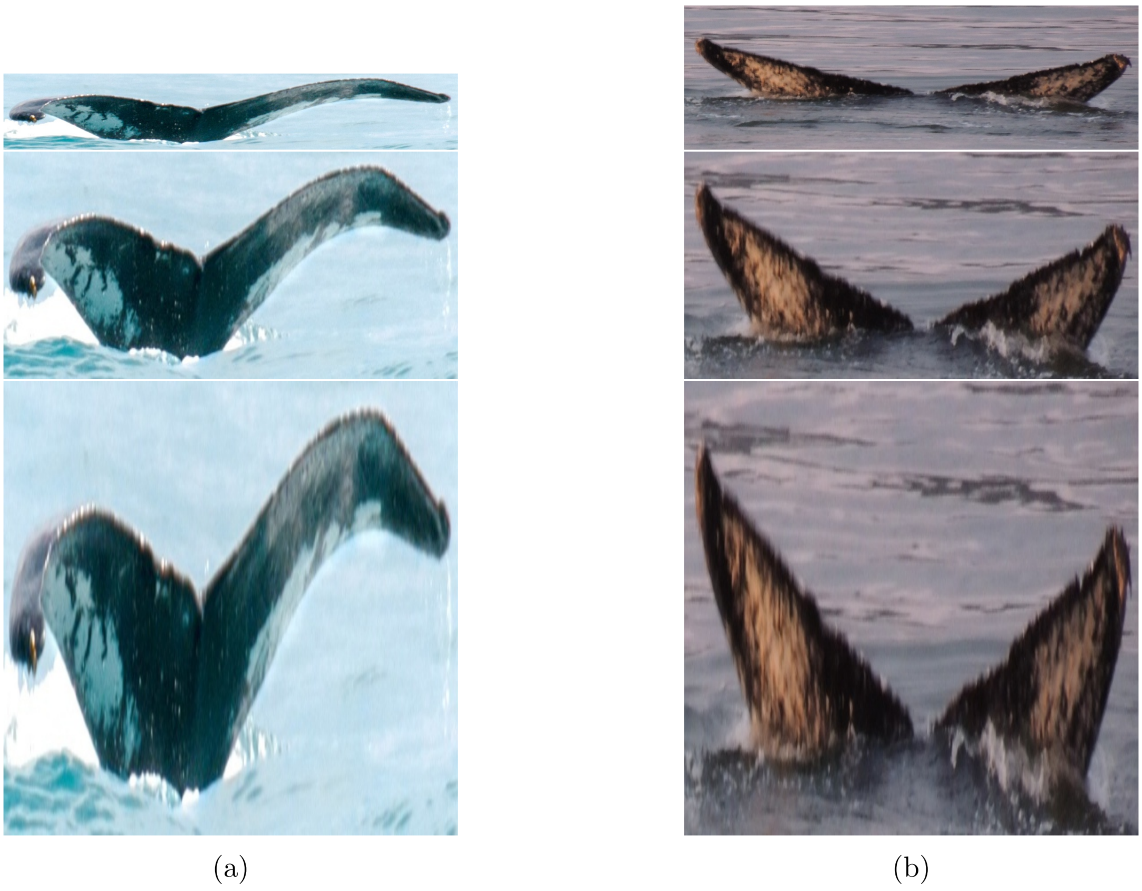


Figure 4.1. Example different aspect ratio benefits. In both examples, the first image (top) is in its original ratio, while the second is resized to $(512, 256)$, and the third (bottom) to $(512, 512)$. All images were first cut using bounding boxes, and then resized. (a) Example where resizing to $(512, 256)$ greatly improves the exposure of the markings, while the 1:1 aspect ratio starts to distort too much the fluke shape. (b) Example which benefits from the $(512, 512)$ image size, due to the even further exposure of the markings without much distortion.

4.1.1 Image Sizes

Instead of choosing a single image size to work with in the model, Shen decided to train all its pipeline using two different image sizes: $(512, 256)$ and $(512, 512)$. He has not given an explanation of why these specific values were selected, but we can see that the exposure of markings on the whale fluke can benefit from different aspect ratios. Figure 4.1 shows examples where both image sizes enhance the visualization of the markings.

4.1.2 Bounding Boxes

Just as the first-placed team did, Shen also used bounding boxes to cut images before resizing and feeding them to the network. The code repository contains the detailed bounding-box method, which is based on Convolutional Neural Networks, but distinct from the nets aimed at segmentation.

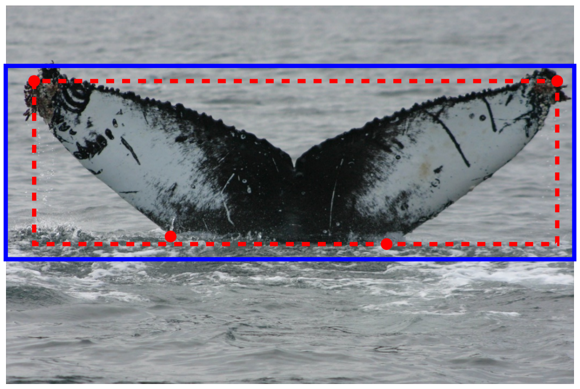


Figure 4.2. Example of the keypoint positions (dots) the network would estimate with the used settings. The dashed rectangle represents the bounding box generated directly using the keypoints, while the solid rectangle is the representation of a bounding box used to cut the images. The latter is just a zoom out (120%) of the former.

Instead, he used a CNN designed for multi-person pose estimation, namely, the Cascaded Pyramid Network (CPN) [CWP+18]. In person pose estimation, networks are intended to output a set of keypoints, which represent each pose-meaningful joint — such as elbows, knees, and shoulders — and body endpoints, including the head, hands, and feet. Besides finding "simple" (visible) keypoints with a *GlobalNet*, CPN introduced a solution to estimate occluded or invisible keypoints more precisely, by using a *RefineNet*, which processes the *GlobalNet*'s feature maps.

GlobalNet uses a backbone network to generate feature maps. Looking for more confident predictions, Shen used two different backbone networks to perform the task: 32x4d SE-ResNeXt-101 and 32x4d SE-ResNeXt-50. With both sets of points available, the final bounding box coordinates are defined as the arithmetic average of the networks' guesses.

For the identification of the fluke, keypoints are set on four extreme points of the tail, as shown in Figure 4.2. A relevant aspect here is the use of a model capable of identifying occluded points, since some flukes in the dataset were indeed occluded by water (see Figure A.5 for an example). However, since this network was only intended to generate the bounding boxes, this capability was not explored further.

4.1.3 Data Augmentation

Following standard practice, Shen also selected a few transformations to apply to images before network processing. The transformations applied are summarized in Table 4.1. Similarly to the first solution authors' choice, he decided to augment the data only virtually, by applying the transformation when an image is loaded and discarding the modified image afterwards.

The next sections discuss each transformation in Table 4.1 in more details.

Transformation set	# of transformations	Probability
Gray scale transform	0	0.5
	1	0.5
Perspective transform without cropping	0	0.5
	1	0.5
Random rotation and shear	1	1
Gaussian noise Gaussian blur Hue and saturation addition Piecewise affine transformation Perspective transform with cropping	0	0.33
	1	0.33
	2	0.33

Table 4.1. Transformation sets selected to augment the training dataset, number of transformations to apply, and probability for each value. For instance, there is a 50% change an image will be transformed into gray scale, then a 50% change it will not have its perspective transformed without cropping, a 100% chance that it will be rotated and sheared, and then a 33% chance of having a single transformation among those listed in the forth row applied to it, *e.g.* Gaussian blur. On each set, the transformations are equally likely to be selected, and are applied in random order.

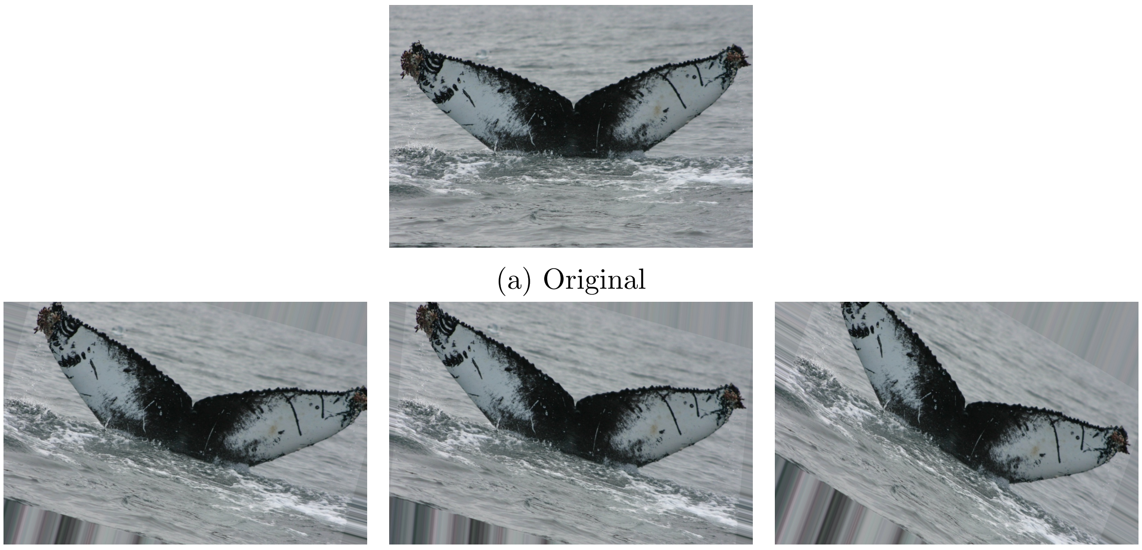
Gray Scale Transform

This transformation consists in literally transforming the image from RGB to gray scale. After the transformation is applied, a tensor with all RGB channels is created (with the same value in all channels), to respect the input format.

Random Rotation and Shear

Rotation and shear are always used to transform training images, and they are applied one after the other. As discussed in Section 3.1.4, rotation is a useful transformation to increase generalization. Shearing has a similar effect, since it is a linear transformation that preserves the image area and modifies the internal angles of the original image. Thus it might be seen as an alternative or a complement to rotation. See Figure 4.3 for a comparative example of these transformations. First, a rotation around the image center is applied, with the angle (in degrees) being randomly chosen from the interval $[-15, 15]$, where a positive angle means clockwise rotation. Then, a shear transformation done upon the result, with x- and y-axis related angles (in degrees) drawn independently from the uniform distribution $U(-15, 15)$ as well.

Instead of filling the empty space created by these transformations with black (*i.e.* zeroed) pixels, Shen used an alternative method which consists in replicating


(a) Original

(b) Rotation (c) Shearing (d) Rotation + Shearing

Figure 4.3. Comparative example of the (b) rotation transformation, (c) shear transformation, and (d) both operations applied together, in contrast to (a) the original image. Fixed angles were selected for both transformations for comparison purpose: 15 degrees for rotation, 5 degrees for x-axis shearing, and 15 degrees for y-axis shearing.

the border pixels across the entire empty extent.

Additive Gaussian Noise

This strategy is essentially the same as that discussed in Section 3.1.4. However, two peculiarities are present in Shen's augmentation. First, he uses a standard deviation sampled from the uniform distribution $U(0, 0.01)$ per image (considering the pixel values lying in the interval $[0, 1]$), which represents a noise of 0% to 1% on average. Second, with a 50% probability the noise values are drawn independently for each color channel, as opposed to a single value for all color channels (as used by Qiao *et al.*). Such a small noise is practically unnoticeable for humans.

Gaussian Blur

Gaussian blur, differently from Gaussian noise, is modeled by a convolution operation, with kernel values being a finite and discrete representation of a two-dimensional Gaussian distribution given by

$$\begin{aligned} G(x, y) &= p(x)p(y) && (4.1) \\ &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2 + (y-\mu)^2}{2\sigma^2}\right), && (4.2) \end{aligned}$$

where $p(t)$ is the one-dimensional Gaussian distribution (Equation 3.2), σ is the standard deviation of the distribution, and (μ, μ) is the highest $G(x, y)$ value coordinate. For this operation, μ is chosen so that its peak is in the central element of the kernel. In other words, the kernel is build in such a way that it is vertically and horizontally symmetrical with respect to the central element, and the values decrease as they depart from the center. In practice, this kind of noise can be found on photos taken when the camera is not properly focused (see Figure A.1 for an example from the Humpback Whale Identification Challenge dataset).

In terms of effects on neural networks, Dodge and Karam show that some state-of-the-art networks are very sensible to this noise, causing most of them to have small changes as first layers propagate to higher layers, and thus possibly changing the prediction result [DK16]. This finding can, possibly, be extended to most Deep Neural Networks found in the literature. Even though deeper networks seem to deal better with this issue, augmenting the data points with this transformation during training is still a reasonable approach.

For this transformation, Shen sampled the standard deviation σ randomly from the interval $[0, 1.5]$. For this σ range, the same kernel size was always used for this data augmentation, which is the minimum kernel size $(5, 5)$ implemented by the library used.

Hue and Saturation Addition

This strategy consists in adding random values of hue and saturation to the image pixels. To achieve this, a transformation from the RGB color space to the HSV color space is done, and a certain value is added to the hue (H) and saturation (S) channels, followed by the inverse transformation.

For Shen's data augmentation, the aforementioned value is drawn from a uniform distribution $U(-5, 5)$. In practice, this means the pixel values will change by the same amount, considering they are lying within the standard 0–255 range. Similarly to Additive Gaussian Noise, the original and resulting image are indistinguishable for humans, but may still improve network robustness.

Piecewise Affine Transformation

An affine transformation can be seen as a linear transformation followed by a translation. It can be shown that this is equivalent to making a series of transformations combining translation, rotation, scaling, and aspect ratio change. This also increases the variability in the test and real world data.

In Shen's solution, piecewise affine transformations were used to create new data points. With this strategy, the image is virtually separated using a regular grid, and only pixels in the same neighborhood are affected similarly. This can create huge distortions when a high number of cells are created. To avoid this unintended effect, a (small) 4×4 grid is used instead. Another parameter used to configure the transformation is the percentage of the image height and width that can be used to make a pixel shift. Shen used a random percentage drawn uniformly from the interval $[1, 3]$. See Figure 4.4 for a comparison of parameters' effects, including an example using the solution configuration.

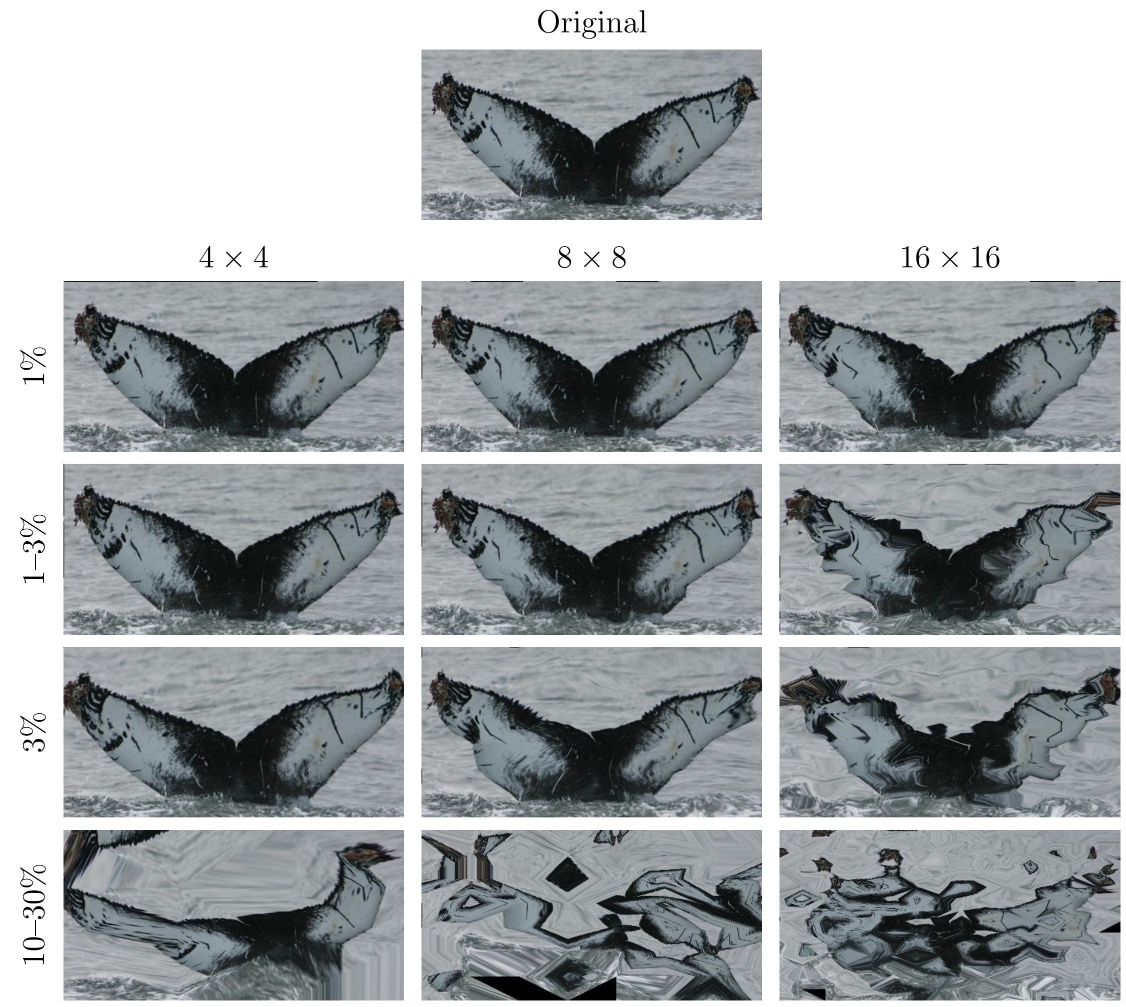


Figure 4.4. Comparison of the effects of changing the grid size *(columns)* and transformation weight percentage *(rows)* for the piecewise affine transformation.

Perspective Transformation

Perspective transformation is an operation which aims at recovering the perspective based on a set of four points (three of which must not be co-linear). For an example, one could have a picture of a sheet of paper on a table taken from a plane not parallel to the table. Then, in order to "fix" the sheet perspective, the sheet's four corner points could be used to make the perspective transformation. The resulting image would look like a photo taken parallel and cropped to fit the sheet size.

However, this transformation is not limited to this. It can also be used to change the perspective of the entire image, which may be cropped right afterwards. Thus, it can introduce another source of variability, without changing the image content drastically. From this point of view, it may be used as a data augmentation transformation.

In Shen's configuration, the distances between each final corner point and its correspondent image corner are set to be at most p percent of the image size. When the image is not cropped, p is drawn (independently for each corner) from a uniform

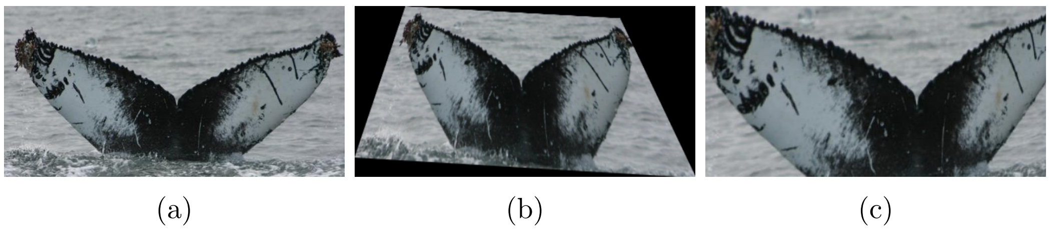


Figure 4.5. Examples of perspective transformation. (a) Original image. (b) Without cropping, with p drawn from $U(0, 25)$. (c) With cropping and $p = 10$.

distribution $U(0, 25)$ in percentage. On the other hand, when cropping is applied, p was drawn from $U(1, 10)$. Moreover, resizing is applied in the sequel in both cases, to keep the same dimensions as before. Figure 4.5 shows examples of results obtained through this transformation with and without cropping.

4.2 Problem Specific Techniques

Following the idea shared on the *Kaggle* Forum, as discussed in Section 3.2, Shen also used the flipped images strategy. In general, the same process is used, and the few differences found are presented in Section 4.3.4.

In addition, pseudo labeling is also used to augment the training dataset. Similarly to the first solution authors, at a point in time when Shen's model reached 0.940 in the public leaderboard, he used it to label 1505 images from the test set (about 19% of all test images), and used these images for training since then.

4.3 Network Architecture

As a recommended practical design process [GBC16], first a baseline model structure was created by Shen, and then constantly enhanced, *e.g.*, by changing loss functions and hyperparameters, until the final structure led him to the top-5 leaderboard positions.

The final model consisted of a *common flow* (described in Section 4.3.1) designed to support different backbone networks attached to it. Different loss functions (shown in Section 4.3.3) were part of this flow, aggregating complementary strategies to improve generalization. In addition, in order to have a broader range of guesses, Shen used three different backbone networks (described in Section 4.3.2) to train several models. Each of these was then used to infer the test image class, and their results were assembled as described in Section 4.4 to produce the final classification.

4.3.1 Common Flow

Possibly to facilitate the development of a solution using model assembling, Shen used a common flow in all his trained models.

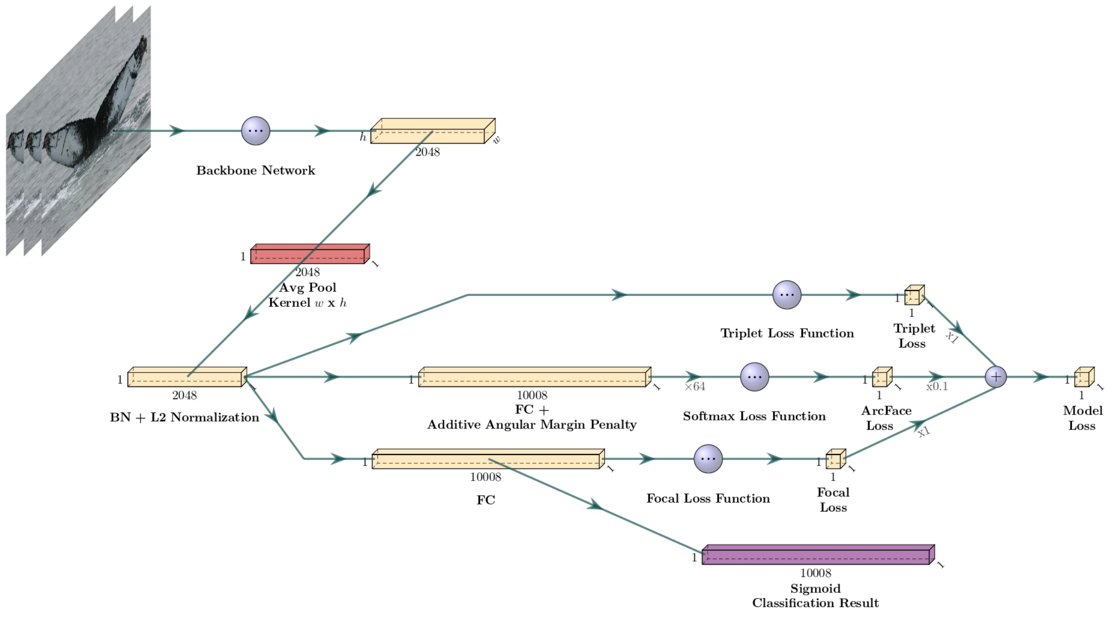


Figure 4.6. Common flow used by Shen to train all models in his solution. In this representation, a single image follows the flow. In practice, however, a batch is required by the Triplet Loss function.

The flow starts with all data treatments covered in Section 4.1. Then, it continues to a backbone network, which processes the batch and returns feature maps. This output is then summarized by average pooling on spatial dimensions, and normalized through batch and Euclidean distance (L2) normalization. At this point, the flow is split into three segments: (a) where Triplet Loss [HBL17] is calculated; (b) where ArcFace Loss [DGXZ19] is generated by the Additive Angle Margin Penalty; and (c) where the output is mapped by a fully connected layer to a classification vector, which is used both to figure out the adapted Focal Loss [LGG$^+$17] and to generate the classification result using a sigmoid function. Finally, losses are combined. An architectural sketch is presented in Figure 4.6.

4.3.2 Backbone Networks

As shown in previous sections, Shen used different backbone networks to have several models trained, which can then be assembled. In his final pipeline, three CNNs are used as backbone nets: ResNet-101 [HZRS16], SE-ResNet-101 [HSS18, HZRS16], and SE-ResNeXt-101 (32×4d) [XGD$^+$17, HSS18]. All these networks have a similar structure, including their depth. Moreover, they have a very similar structure with the CNN shown in Chapter 3: SENet-154. As one might recall, SENet is actually a slightly modified ResNeXt-101, which adopts block stacking and SE blocks.

However, there are differences among these three CNNs regarding internal block architecture and strategies used in their paths. We present next the overall structure of each of these networks, referring to architectural sketches from Chapter 3 whenever appropriate.

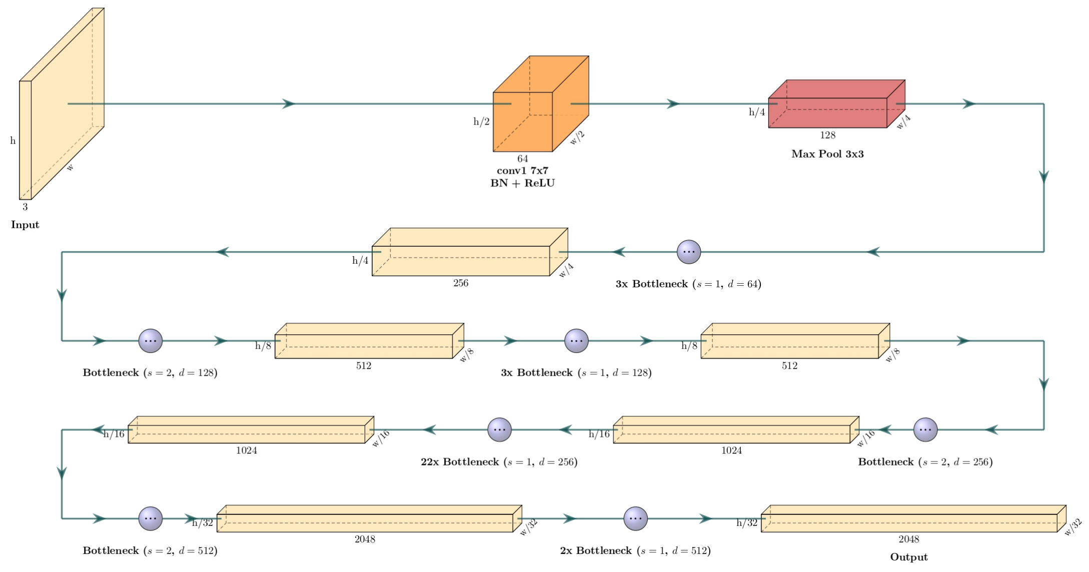

h
w
3
Input
h/2
w/2
64
conv1 7x7
BN + ReLU
h/4
w/4
128
Max Pool 3x3
h/4
w/4
256
3x Bottleneck ($s = 1,\ d = 64$)
Bottleneck ($s = 2,\ d = 128$)
h/8
w/8
512
3x Bottleneck ($s = 1,\ d = 128$)
h/8
w/8
512
h/16
w/16
1024
22x Bottleneck ($s = 1,\ d = 256$)
h/16
w/16
1024
Bottleneck ($s = 2,\ d = 256$)
Bottleneck ($s = 2,\ d = 512$)
h/32
w/32
2048
2x Bottleneck ($s = 1,\ d = 512$)
h/32
w/32
2048
Output

Figure 4.7. Overall structure of ResNet-101, SE-ResNet-101 and SE-ResNeXt-101. The differences among these networks are concentrated in the bottleneck level.

ResNet-101

Residual Networks were first proposed by He *et al.*, and were designed to solve the problem of degradation due to large network depth [HZRS16]. The key idea is to choose a residual function to be learned by the network blocks instead of the direct mapping. The residue $\mathcal{F}(X)$ is the difference between the target mapping $Y(X)$ and the identity mapping $I(X) = X$ of the input features X. Mathematically, this can be described as

$$Y = \mathcal{F}(X) + X. \tag{4.3}$$

Thus, $\mathcal{F}$ is the (residual) function to be learned by the network instead of Y directly. This formulation results in more stability, and consequently allows greater depths without the degradation issue [HZRS16].

ResNet-101 was one of the first models proposed in the work of He *et al.* Its overall structure is shown in Figure 4.7. Other Residual Networks used by Shen, which are described next, use the same overall structure, except for slight differences in bottleneck design. Figure 4.8 shows a comparative sketch of the bottlenecks of the three residual networks used by Shen.

SE-ResNet-101

Aiming at improving deep convolutional networks, Hu *et al.* [HSS18] proposed the Squeeze-and-Excitation blocks, as described in Section 3.3.1. One of the networks resulting from their contribution is an adaptation of ResNet-101: SE-ResNet-101. This adaptation includes SE Blocks in the network bottleneck, as shown in Figure 4.8. Moreover, in the implementation code used by Shen, the order of the strided convolution at the beginning of the block is also different.

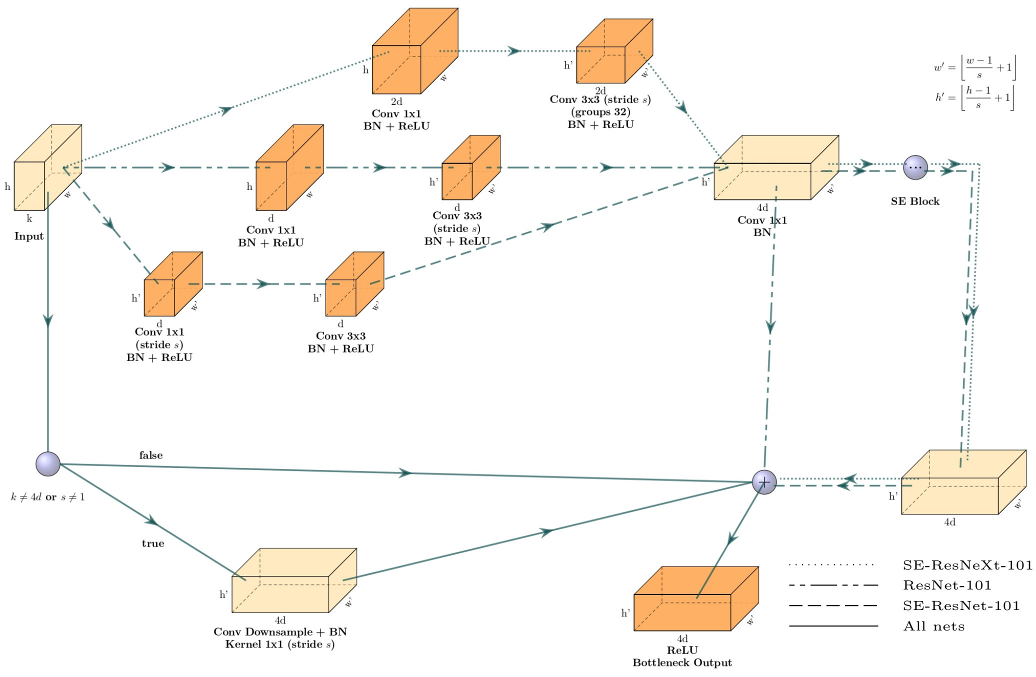


Figure 4.8. Comparative representation of the bottleneck used by ResNet-101, SE-ResNet-101, and SE-ResNeXt-101 (32×4d). The SE Block, which is part of the SE-ResNet-101 and SE-ResNeXt-101, is exactly the same shown in Figure 3.8.

SE-ResNeXt-101 (32×4d)

Another network benefiting from the incorporation of SE Blocks is ResNeXt. This network is a modification of ResNet, proposed by Xie *et al.* [XGD+17], which includes another adaptation to the original network flow. Using the idea of splitting, transforming, and aggregating, they proposed an aggregated transformation, which is given by

$$\mathcal{F}(X) = \sum_{i=1}^{C} \mathcal{T}_i(X), \tag{4.4}$$

where X is the transformation input, C is the cardinality (size) of the set of transformations, and the $\mathcal{T}_i$ can be arbitrary functions. Applying this idea along with the residual learning framework, the block turns out to be given by

$$\begin{aligned} Y &= X + \mathcal{F}(X) \qquad (4.5) \\ &= X + \sum_{i=1}^{C} \mathcal{T}_i(X), \qquad (4.6) \end{aligned}$$

where Y is the block output. Aggregated transformations can be seen as introducing another dimension to the network besides width and depth. Xie *et al.* show that this new dimension increases performance [XGD+17].

For ResNeXt, all $\mathcal{T}_i$ have the same topology. In this case, $\mathcal{F}$ can be implemented as a grouped convolution. This is how SE-ResNeXt-101 was implemented in Shen's solution, as represented in Figure 4.8. In the ResNeXt used by him, the cardinality

C is set to 32, while the block output's depth is $4d$, where d is a hyperparameter with different values for each block stack (see Figure 4.7).

Another distinctive characteristic of this network is the higher number of feature maps generated in the bottleneck first convolutions, which is twice the others. This number of features is similar to the one in SENet-154, discussed in Chapter 3.

4.3.3 Loss Functions

In Shen's final model design, four loss functions are combined to generate the model loss: Triplet Loss [HBL17], ArcFace Loss [DGXZ19], CosFace Loss [WWZ$^+$18, WCLL18], and an adaptation of Focal Loss [LGG$^+$17]. Each of these functions is applied at a different branch of operations, as shown in Figure 4.6. We discuss each of them in the next sections.

Triplet Loss

Following a similar approach used by Qiao *et al.*, Shen also used the Triplet Loss function in his pipeline, despite not considering it a key aspect of his solution. See Section 3.3.3 for a brief explanation of this loss function.

A difference between Shen's application and Qiao *et al.*'s relies on the hard sampling strategy. While in the first-placed solution classes were chosen randomly, along with random positive and negative examples, the second-placed solution used the technique specified by Hermans *et al.* [HBL17], *i.e.* a batch is created using P classes with K examples each. In Shen's implementation, K was set to 4, and P was such that the batch size matched a previously specified value. Moreover, a fixed ratio of around 25% of *new whales* is always present in the sampled batches.

ArcFace Loss

Here we present in more details the ArcFace Loss [DGXZ19], which is part of a branch of the common flow, and is considered the key aspect of the solution by the author, as stated in a response to a competitor [She19a].

Softmax Loss is widely used in classification tasks using neural networks. In its standard form, it is given by

$$\mathcal{L}_{std} = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{\exp\left(W_{y_i}^T x_i + b_{y_i}\right)}{\sum_{j=1}^{n}\exp\left(W_j^T x_i + b_j\right)}\right), \tag{4.7}$$

where N is the batch size, W and b are respectively the weight matrix and the bias vector of the last linear layer, y_i and x_i are respectively the correct label and the feature vector for the i-th sample, and n is the number of classes. However, this standard form is shown to have a poor power of discrimination [WWZ$^+$18]. Aiming at solving this issue, different adaptations have been proposed to the standard Softmax Loss. Most of them use the idea of normalizing both W and x_i using the Euclidean norm and setting the biases to zero, allowing a geometric interpretation of the inner product between W_j and x_i as $\cos\theta_j$, *i.e.* the cosine of the angle formed by the vectors in the hyperspace. Moreover, a cosine re-scaling factor s is usually used, as first proposed by Wang *et al.* [WXCY17].

ArcFace Loss is one of these adaptations, proposed by Deng *et al.* [DGXZ19]. Their idea is essentially to introduce an additive angular margin m to the argument of the cosine, yielding the following mathematical expression for the loss:

$$\mathcal{L}_{arc} = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{\exp\left(s\cos(\theta_{y_i}+m)\right)}{\exp\left(s\cos(\theta_{y_i}+m)\right)+\sum_{j=1,j\neq y_i}^{n}\exp\left(s\cos\theta_j\right)}\right). \tag{4.8}$$

This enlarges the decision boundary associated with the classes and increases discriminatory power. For an analysis of loss function and intuitive examples of the effect of the margin, see Deng *et al.*'s work [DGXZ19].

In Shen's implementation, the margin parameter is configured as suggested by the authors, *i.e.* $m = 0.5$, while the scaling factor is set to $s = 64$. However, in his solution, this loss function is only applied when the $\theta_j \in [0, \pi]$. When θ_j is outside that interval, CosFace Loss is used instead, which is described in the next section.

CosFace Loss

CosFace Loss, also called Additive Margin Softmax Loss, is another adaptation proposed to enhance the Softmax Loss [WWZ+18, WCLL18]. Following a similar approach to ArcFace's, this loss introduces a margin parameter also aimed at enlarging the decision boundary for classes. However, this margin is added in the cosine space instead. See Deng *et al.*'s work to a comparative analysis including these different margin penalties [DGXZ19].

When this loss is applied in Shen's solution, the additive margin parameter is set to $m\sin(m)$, where m is the margin configured for ArcFace Loss. Since $m = 0.5$ for the angular additive margin, $m\sin(m) \approx 0.2397$ is the numeric value used for the CosFace margin parameter, which is relatively close to the authors' recommended value of 0.35.

Focal Loss Adaptation

Inspired by the Focal Loss idea of weighting the loss function output differently based on the confidence of the network and the correctness of the prediction, Shen created an adaptation of this loss function. This modified function also includes the Binary Cross-Entropy Loss, described in Section 3.3.3. In order to describe it, first some auxiliary functions must be introduced, namely

$$h(x) = \begin{cases} 0, & \text{if } x > 0; \\ |x|, & \text{otherwise,} \end{cases} \tag{4.9}$$

$$\zeta(x) = \log(1 + e^x), \tag{4.10}$$

and

$$X^*_{j,i} = \begin{cases} -X_{j,i}, & \text{if } i = y_j; \\ X_{j,i}, & \text{otherwise,} \end{cases} \tag{4.11}$$

where $X_{j,i}$ is the fully connected layer output shown in Figure 4.6, with the first index representing the batch example, the second index representing the i-th class, and y_j being the j-th example label. Shen's adapted function is given by

$$\mathcal{L} = \mathcal{L}_{BCE} + \frac{1}{Cn} \sum_{j=1}^{C} \sum_{i=1}^{n} \sigma(X^*_{j,i})^{\gamma} \left[h(X^*_{j,i}) + \zeta(-|X_{j,i}|) \right]. \tag{4.12}$$

where C and n are, respectively, the batch size and the number of classes, and γ is a hyperparameter set at 2.

4.3.4 Inference Phase

As shown in Figure 4.6, the inference phase follows the path that leads to the classification output. This path processes both non-flipped and flipped test images, resulting in two classification results by each model. Both values are combined later on, as shown in Section 4.4. When a test image is flipped, only the vector entries regarding flipped classes are used. This way, flipped images are classified, in the end, as the non-flipped class corresponding to the inferred flipped class.

4.3.5 Training Procedure

Following the same strategy used by Qiao *et al.*, Shen used transfer learning and started with models pre-trained with the ImageNet dataset [RDS+15]. Every model was then trained to specialize in the task of identifying whales by their tails. To accomplish this, the first two backbone network layers are frozen, while the others (most of the net) are free to learn, specializing to solve the task at hand.

In terms of the flow used for model training, an approach differing from the first-placed solution was used, though. In the present solution, all networks are trained separately, always for a hundred epochs. During this phase, the learning rate is automatically updated, as discussed below in the section on hyperparameters.

Parameter Initialization

By using transfer learning, practically all model parameters are already set at start up, since Shen's common flow does not extend much the network flow through convolution or other parametric operations. For Batch Normalization, shown in Figure 4.6, neither the mean $\mu = 0$ nor the standard deviation $\sigma = 1$ are set as parameters to be learned.

The fully connected layer on the ArcFace Loss path is initialized with weight and bias parameter values drawn from the uniform distribution $U(-\sqrt{k}, \sqrt{k})$, with $k = 1/C_{in}$, where C_{in} is the number of input channels (2048). On the adapted Focal Loss path, the weight matrix is initialized with values drawn from the uniform distribution $U(-1, 1)$, followed by an L2 normalization on the class dimension.

Hyperparameters

Besides the hyperparameters already discussed throughout this chapter, such as loss margins, a few additional hyperparameters were configured for the final model.

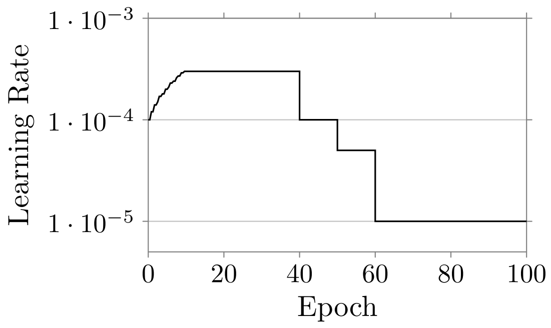


Figure 4.9. Learning rate warm-up schedule.

Optimization Algorithm Parameters. The Adam optimization algorithm [KB14] is used, with exponential decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and weight decay $\alpha = 2 \cdot 10^{-4}$.

Learning Rate. The base learning rate is set to $3 \cdot 10^{-4}$, and is constantly updated during the training as described in the next section.

Warm-up Learning Rate

Learning rate warm-up is a heuristic for adapting the learning rate during training. When a significantly large batch size is used to train a network, a poor learning progress is empirically identified. However, this heuristic is shown to help prevent large and unstable changes in the fully connected layers of the network when a large batch size is used, thus leading to a successful training [GKXS18].

Shen decided to include this strategy in his solution, and defined the learning rate update schedule based on the training epoch shown in Figure 4.9.

4.4 Model Ensemble

As mentioned in previous sections, the final classification is given by an ensemble of models. Each model varies in terms of (i) the backbone network used, (ii) the input image size, and (iii) the usage of pseudo labels. Table 4.2 shows each combination more precisely. Moreover, for each model, two classification outputs are generated: one for original test images and other for flipped test images. Both classification results are used for creating the final classification, which means that 20 different predictions are combined. Possibly to reduce the computational burden, Shen had all classifications used in the assembling procedure contain only the top-5 high confidence predictions.

4.5 Reproducing Solution Training

All procedures and steps needed to reproduce Shen's solution are available on the *GitHub* source code repository [She19b]. Along with the code itself, bounding boxes and pseudo labels used are also available. We followed strictly the process of training each model, which essentially translates into following what is specified in Table 4.2.

Model	Backbone Net	Image Size	Pseudo Labels	Weight
(a)	ResNet-101	512 × 256	-	1
(b)			✓	2
(c)		512 × 512	-	1
(d)			✓	2
(e)	SE-ResNet-101	512 × 256	-	1
(f)			✓	2
(g)		512 × 512	-	1
(h)			✓	2
(i)	SE-ResNeXt-101	512 × 256	-	1
(j)		512 × 512	✓	2

Table 4.2. Ensemble models and their respective vote weight.

Since our hardware configuration probably differed from his, for each trained model, we adapted the batch size to fit our resources. Memory was the main factor impacting the batch size, because larger image sizes imply more parameters in the network (increasing memory usage). Table 4.3 shows the maximum validation precision, batch size, and training time for each trained model.

Finally, our reproduction official score is shown in Table 4.4. It's notable that a very similar result was achieved, indicating a good reproduction of the neural network training and post-processing steps.

4.6 Conclusion

In order to achieve a good classification, this solution incorporated standard practices to transform the data, reducing the impact of variability on the input, and used different image sizes to train several deep residual neural network models, which formed an ensemble. Each of the trained models shared a common flow, which included different loss functions, some of which are based on ideas from the fields of face recognition and person re-identification. The final result is built using 20 classification results voted by 10 trained models. In our reproduction, the performance achieved was very close to the authors', with less than 0.005 in absolute difference.

Model	Validation		Batch Size	Training Time
	MAP@5	Top@1		
(a)	0.943	0.917	64	31h00
(b)	0.947	0.920	64	29h30
(c)	0.926	0.885	16	117h00
(d)	0.932	0.891	16	111h30
(e)	0.934	0.899	64	37h30
(f)	0.942	0.910	64	35h30
(g)	0.924	0.887	32	73h00
(h)	0.934	0.900	32	34h00
(i)	0.940	0.909	64	14h30
(j)	0.941	0.913	32	80h30

Table 4.3. Partial training results for each model. The training time is an approximation of the time taken to achieve the highest validation MAP@5, which occurred in different epochs for each model.

Model	Leaderboard score	
	Public	Private
Shen's	0.97359	0.97208
Ours	0.97150	0.96910

Table 4.4. Official results on public and private leaderboards on Kaggle Platform for Shen's final submission and our submission.

Chapter 5

Third Place Solution

This solution was developed by Jinmo Park, and achieved the second place on the public leaderboard, and the third place on the private leaderboard. A brief summary of the solution and its progress during the competition is available on a post on the *Kaggle* Forum [Par19a]. The source code is available on the *GitHub* platform, along with all configuration files and extra resources necessary to reproduce this solution [Par19b].

Park's solution also used neural networks both to generate bounding boxes and landmarks as well as to identify the whales by their tails. However, differently from the top-2 solutions, the classification learned directly by the network is not used during the inference phase. Instead, the set of features discovered for each identity in the dataset are compared to the features of each test image during the inference phase, in order to decide whether both images belong to the same whale.

Many other techniques are used in Park's solution. We will explore them, with emphasis on the points not presented in Chapters 3 and 4. We will also present the overall configuration. Thus, in Section 5.1, we show how Park dealt with the data. Then, we present some special tactics applied on Park's solution in Section 5.2. In the sequel, we describe in Section 5.3 the neural network architecture used, and how inference is done using the learned features in Section 5.4. Finally, we present our results in reproducing the solution training and test.

5.1 Data Treatment

This solution followed a similar strategy compared with the top two solutions: standardize the image format and apply several transformations onto the images. For standardizing input images, bounding boxes and landmarks are created for each image, and are used to crop and align the whale fluke, as shown in Section 5.1.1. The other transformations are applied aiming at data augmentation and regularization. These are described in Section 5.1.2.

5.1.1 Bounding Boxes and Alignment

As the network's standard input image format, Park decided to have images cropped in such a way that the resulting image had mostly the whale's fluke on it. This way,

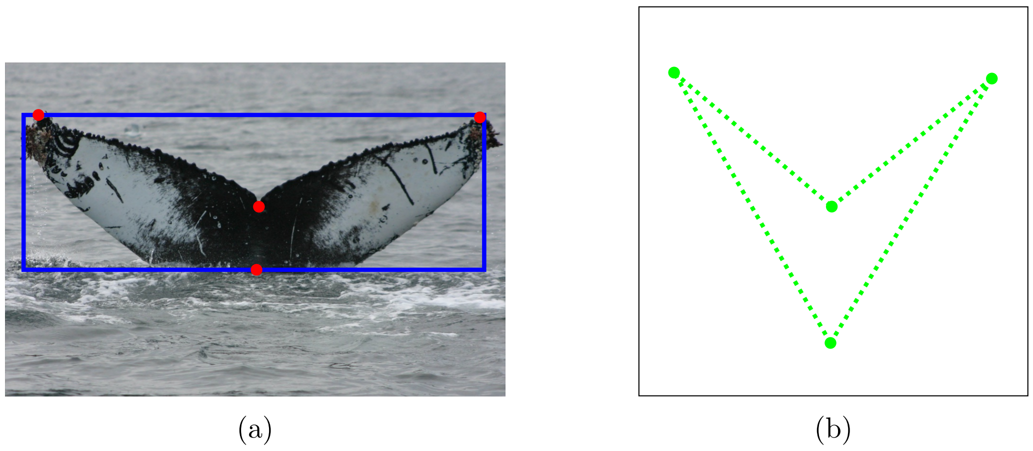


Figure 5.1. Example of bounding box and landmarks used by Park. *(a)* Predicted bounding box (solid line) and landmarks (dots) for an image in dataset. *(b)* Alignment point positions after cropping and resizing the image. The dotted lines indicate fluke limits (not used in the code — shown here for illustrative purposes only).

noisy background information could be more easily ignored by the network during the inference process. Moreover, an alignment technique was thought at first to be always applied to the images, aiming at a standardizing the fluke's position. In order to "align" the fluke, some keypoints (landmarks) were chosen by Park to use as reference. This information, however, was not part of the original dataset. Thus, another task consisting in predicting both bounding boxes and landmarks had to be pursued. With indirect help from another competitor, namely Johnson, who made available a set of hand-labeled keypoints for a thousand images [Joh19], Park was able to create a deep learning solution to detect these landmarks. This deep learning solution also predicted the bounding boxes, trained with the data provided by Osmulski [Osm19]. The bounding boxes and fluke keypoints used by Park are illustrated in Figure 5.1.

Following image cropping, a resize operation is performed, standardizing the input image size to $(320, 320)$. After some attempts to tackle the problem, Park noticed that the alignment operation could have been worsening the results, due to inaccurate prediction of landmarks and bounding boxes. Thus, he decided to align each training example with a probability of 0.5. In contrast, all test examples are always aligned. This slightly increased his score. Therefore, alignment can also be seen as a form of data augmentation in his solution, since it is not always applied to the training images. Other data augmentation strategies are discussed in the next section.

5.1.2 Data Augmentation

Some transformations were chosen by Park for his strategy to augment the data (see Table 5.1). Most of them are simple and used mainly to increase variability, aiming at a better generalization. Since some similar transformations have already been discussed in previous chapters, we focus here on the transformations not previously seen, referring to the others when necessary.

Transformation		Probability
Blur	Average	0.5
	Motion	0.5
Random addition		1
Random multiply		1
Random scale Random translation Random shear Random rotation		0.5
Random gray scale		0.5

Table 5.1. Data transformations used by Park.

Average and Motion Blur

Like Gaussian Blur, discussed in Section 4.1.3, average and motion blur are applied using kernels, in order to affect the neighboring pixels only. In average blur, the kernel calculates the average value of the region and replaces the central element. In Park's solution, $3x3$ kernels are used for this operation. Motion blur, on the other hand, makes a transformation that simulates the distortion on the image due to abrupt motion when taking a photo. For this type of blur in his solution, a random kernel size is drawn from uniform distribution $U(3, 5)$, which causes an almost imperceptible visual effect due to the high resolution of the images.

Random Addition and Multiply

Random addition is an arithmetically simple transformation which adds a random value to all pixel values. In Park's augmentation, the random value was chosen uniformly from the interval $[-10, 10]$ per image. With a probability of 0.5, random values were chosen separately for each color channel. Random multiply is analogous to addition, but here the image tensor is scaled by the sampled value. In this solution, the scaling factor was drawn from the uniform distribution $U(0.9, 1.1)$, and also had a 0.5 probability of having different values sampled per channel.

Random Scale, Translation, Shear and Rotation

Scaling, translation, shearing and rotation are all affine transformations, and they are all applied together. Scaling refers to the change of the original image spatial dimensions by some factor, followed by a filling operation when the transformation shrinks the image, in order to preserve the tensor dimensions. Park chose to use independent scaling factors on the spatial dimensions, both drawn from the distribution $U(0.9, 1.1)$.

In Park's solution, translation is based on a random percentage of the image

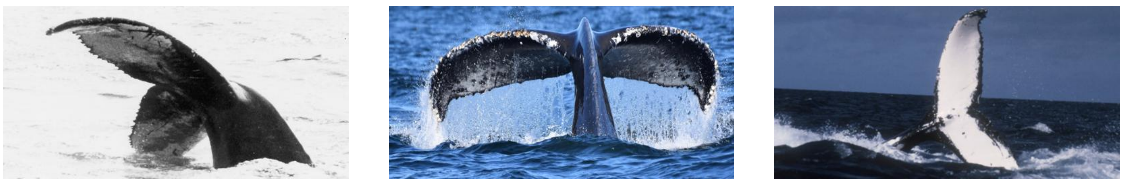

(a) 90 degrees (b) 180 degrees (c) 270 degrees

Figure 5.2. Examples of image rotation due to different fluke position on the photograph. Rotation angles are clockwise.

dimensions. For both width and height, a random value in drawn from the distribution $U(-0.05, 0.05)$. Pixel values undefined due to the translation are zeroed, while the pixels outside the image are ignored.

The last two affine transformations, shearing and rotation, are illustrated in Section 4.1.3. Here both operation have their angles drawn uniformly from the interval $[-10, 10]$ (in degrees), and have the empty tensor entries zeroed instead of replicated.

Random Gray Scaling

Random gray scaling is a transformation that composes the original image and its grayscale version using a random percentage α of pixel's exclusion from the original image. Thus, when $\alpha = 0$, the image is not changed at all. On the other hand, when $\alpha = 1$, the resulting image is entirely in gray scale. Park used an exclusion probability drawn from the distribution $U(0.8, 1)$. Therefore, original colored images are mostly converted to gray scale when the operation is applied.

5.2 Special Techniques

Exploring the data characteristics, this solution groups mislabeled whales — including some *new whales* —, rotates some images, uses a classification leak, and flips the images to create more identities. Next we briefly discuss each of these tactics.

5.2.1 Image Rotation

In order to keep the alignment consistent, Park detected some images which originally did not have their landmarks in a similar arrangement as the standard position shown in Figure 5.1. Therefore, they are rotated to better position the flukes, and to improve the alignment result. It worth noting this transformation is not exploited on the landmark prediction task, though. See Figure 5.2 for some examples of images chosen to be always rotated during training the fluke identifier.

5.2.2 Class Grouping

As also discovered by other participants, the competition dataset contained some mislabeled images [Mok19]. This mislabeling consisted of identifiable whales with some of their images labeled as *new whale*, and different IDs for images of the same whale. To solve the multi-label issue, Park created new IDs to unify different labels for the affected whales. Moreover, images classified as *new whales*, but with multiple images in the dataset, also had new IDs assigned (although, in the end, it were classified as the general *new whale* class).

As a part of a post-processing phase, Park exploited the image sizes to decide which of the duplicate classes should be the first predicted class. This is done so that the first predicted class is always the one that has an image whose size is equal to the size of the test image, if there is any.

5.2.3 Classification Leak

A classification leak occurs when a competitor already knows the label of one or more test images. Park used a leak published by Wang — member of the first-placed team [Wan19] — although not in a way affecting the task itself, when seen as an application to be used afterwards. This leak consisted of labels for test images from the featured competition which were also available in the previous, playground competition. He used this information to replace the top-1 class for each test image involved.

5.2.4 Flipped Images

Similarly to the first- and second-placed teams, Park used the technique of flipping images to create fake identifiable whales. For details about this strategy, refer to Section 3.2.1.

5.3 Network Architecture

The architecture developed to identify the whales has a few modifications in terms of the default setting proposed in the literature. The model uses a Densely Connected Network as the backbone network [HLVDMW17], which is discussed in Section 5.3.1.

Figure 5.3 shows an architectural sketch of the network. As it can be noticed, the backbone network is used to generate a set of features, which are later used to identify the whales, in a process discussed in Section 5.4. To generate the model loss during training, a pipeline consisting of batch normalization, dropout, a fully connected layer and another batch normalization is used to generate the feature vector, where the ArcFace approach (discussed in Section 5.3.2) is used. During inference, only the feature vector is used, as described in Section 5.4.

In Section 5.3.3, we present parameter settings and the solution's training procedure.

5.3.1 Backbone Network

In order to compute high-level features from the input image to be used in the identification task, a Densely Connected Convolutional Network is incorporated to the solution; more precisely, the DenseNet-BC-121.

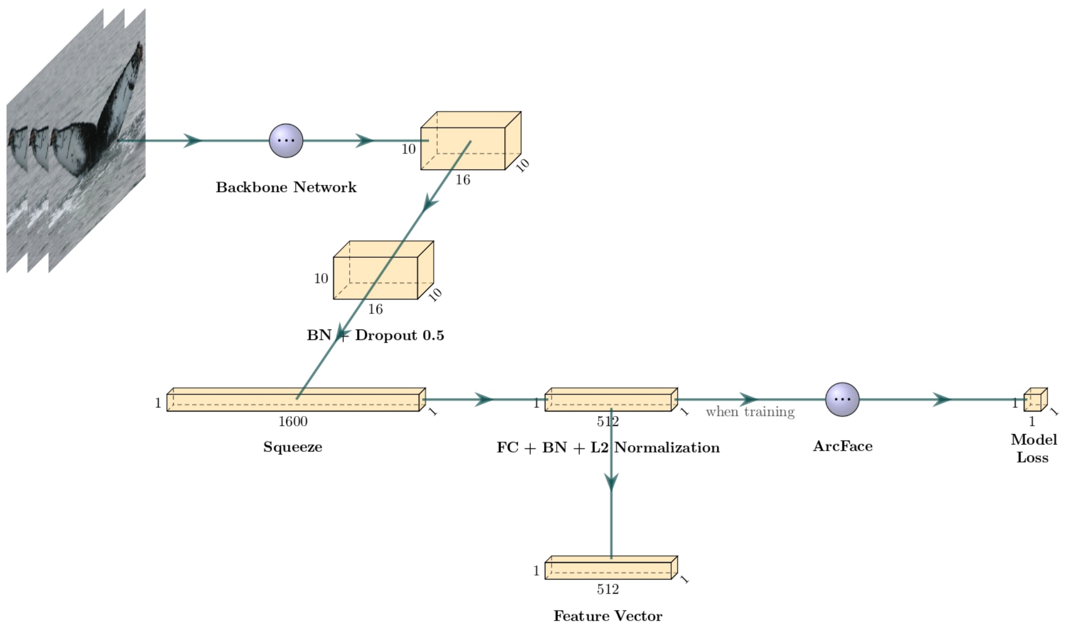


Figure 5.3. Network design used by Park to generate embeddings (feature vectors) for the whales' flukes.

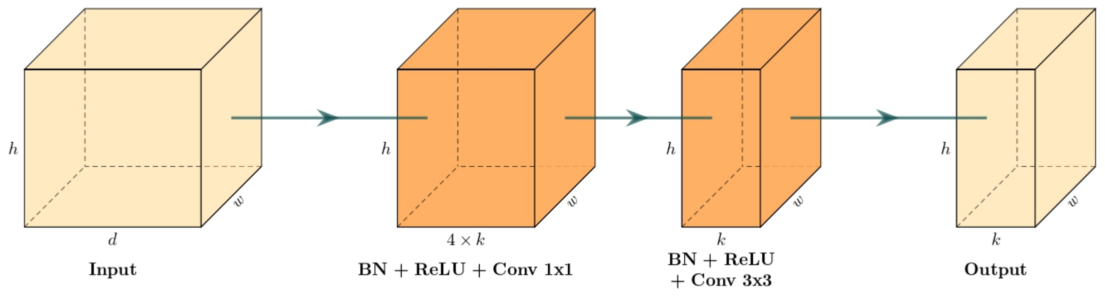
DenseNets were first presented by Huang *et al.* as an architecture that exploits even further the idea of skipping connections introduced by He *et al.* with Residual Networks [HLVDMW17, HZRS16]. For Densely Connected Networks, instead of having a single skipping or residual connection, all layers are directly connected with previous layers. This connection is done through the concatenation of features, in contrast to the addition operation of Residual Networks. Huang *et al.* show many benefits this architecture can bring, including better accuracy, parameter efficiency, and less propensity to overfitting.

Next we present an overview of the complete DenseNet-121 architecture with this solution configuration. The variant DenseNet-BC uses both bottleneck (B) and compression (C) operations. For more details about other variants of Densely Connected Networks, see Huang *et al.*'s work [HLVDMW17].

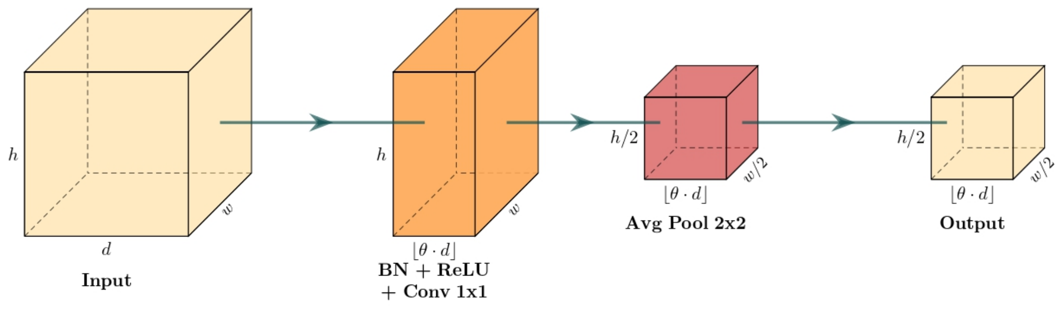
Dense Blocks and Transition Layers

Dense blocks are the fundamental building blocks of a DenseNet. A dense block is where the dense connectivity of layers takes place, building the accumulated knowledge of the network. Each dense block has a similar structure: several composite operations followed by a transition layer. As shown in Figure 5.4a, the composite operations are defined as 1x1 convolutions, with batch normalization (BN) and ReLU activation — which participates on the Bottleneck variant only, since it is responsible for reducing the number of feature maps —, followed by another convolution (3x3) with BN and ReLU.

Transition layers are responsible for reducing the spatial dimensions of the features generated by the dense blocks, and for summarizing the information needed by upcoming blocks. Moreover, since this implementation uses compression, a ra-



(a) Composite operation.


h
d
w
Input
h
$\lfloor \theta \cdot d \rfloor$
w
BN + ReLU
+ Conv 1x1
h/2
$\lfloor \theta \cdot d \rfloor$
w/2
Avg Pool 2x2
h/2
$\lfloor \theta \cdot d \rfloor$
w/2
Output

(b) Transition layer.

Figure 5.4. The two fundamental layers of the DenseNet architecture. Both blocks are illustrated with an arbitrarily-sized input feature map. *(a)* Composite operation layer. Parameter k is the growth rate (set to 32). *(b)* Transition layer used between dense blocks in the DenseNet-BC-121. Parameter θ is the compression ratio mentioned in the text.

tio $\theta = 0.5$ of feature map reduction is also present, possibly removing redundant information the block creates by reducing the number of feature maps. Figure 5.4b shows a transition layer configuration.

Overall Structure

Roughly speaking, a DenseNet is built using dense blocks, interspersed with transition layers. For a 121-layer DenseNet, four dense blocks are used, with 6, 12, 24, and 16 composite operations, respectively. Before starting the dense block pipeline, a convolution (7x7) is applied, followed by a (max) pooling. A sketch of this architecture is shown in Figure 5.5. Note that the last dense net does not have a transition layer. Instead, batch normalization is performed. In a conventional DenseNet, a linear layer with softmax would follow max pooling, in order to generate the classification result. However, this part is adapted by Park, and the flow follows the scheme specified in Figure 5.3.

5.3.2 Loss Function

Similarly to the second-placed solution, Park also modeled his solution with the ArcFace approach [DGXZ19], described in Section 4.3.3. The same strategy was

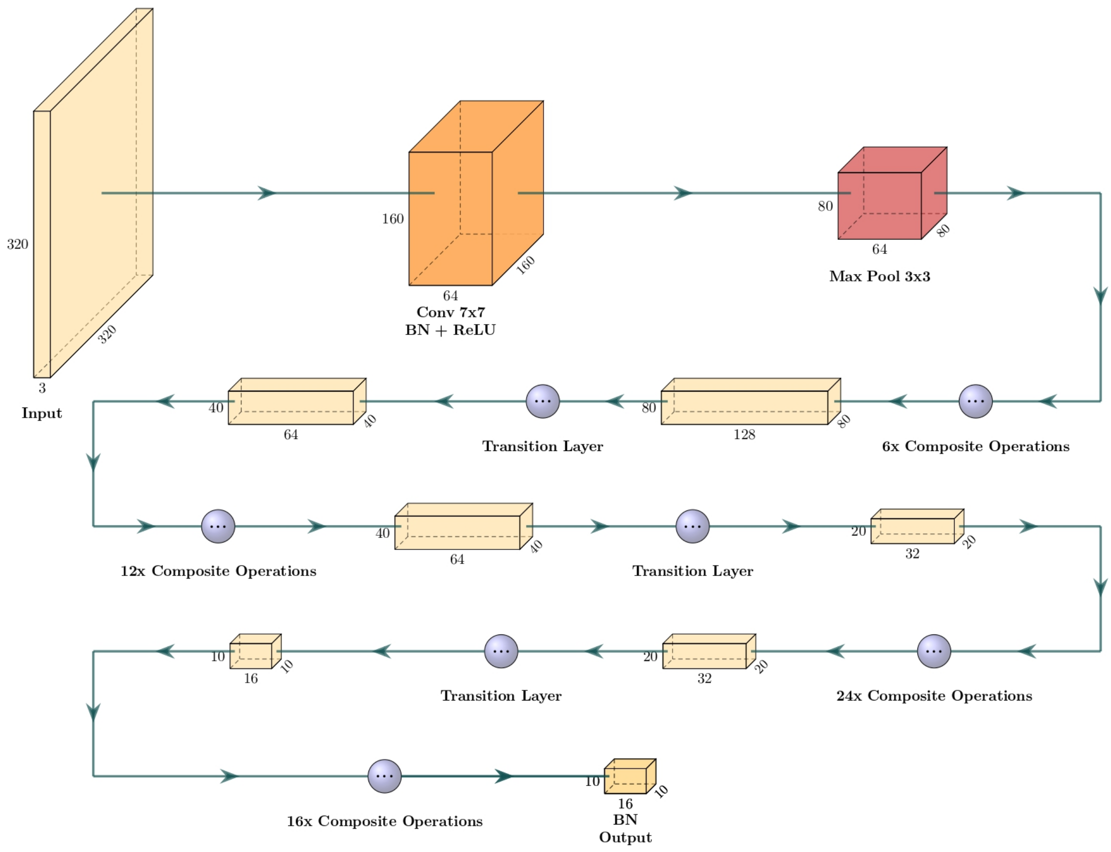


Figure 5.5. DenseNet-121 structure. The classification layer is not represented in this sketch, since the third place solution had another identification strategy.

used in terms of the CosFace usage for angular values from interval $[0, \pi]$. However, as shown in the previous section, this loss function is the only one contributing to the model final loss.

5.3.3 Training Procedure

The backbone network is imported indirectly from the *PyTorch* implementation, and it uses the ImageNet pre-trained parameters [Cad17, RDS+15]. Differently from the top two solutions, Park does not freeze any network layer during the training. However, a scheduler is used to update the learning rate, impacting on the learning process. Next, a description of parameter initialization, hyperparameter configuration, and scheduler for this network are presented. Finally, we briefly comment on the Stochastic Weight Averaging strategy [IPG+18], also a feature of this solution.

Parameter Initialization

Besides the backbone network parameters — which are given by transfer learning —, Park initialized the batch normalization and the fully connected layer in Figure 5.3 with the default configuration from PyTorch. In other words, for the normalization, the learnable mean γ is set to 1 and standard deviation β to 0; for the FC layer, weights and biases are uniformly initialized with values from $U(-\sqrt{k}, \sqrt{k})$, with $k = 1/C_{in}$, where C_{in} is the number of input features. On the other hand, the fully connected layer part of the ArcFace approach is initialized using the Glorot and Bengio method [GB10].

Hyperparameters

Besides the hyperparameters already specified throughout this chapter, Park configured the others as follows.

Optimization Algorithm Parameters. The Adam algorithm is used with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight decay $\alpha = 5 \times 10^{-4}$ [KB14].

Learning Rate. At start-up, it is set to $\alpha = 5 \times 10^{-4}$ on most models (see Section 5.5 for other configurations). Moreover, a scheduler is used to tune this value during training. This tuning is described in the next section.

Multi-Step Scheduler

The multi-step scheduler changes the learning rate at some points in the training procedure, specified in terms of epochs, usually called milestones. This change takes place by a constant re-scaling with factor γ, which diminishes the learning rate. In Park's solution, $\gamma = 0.5$, and the milestones are hyperparameters that depend on the model being trained. As shown in Section 5.5, all models use two milestones, with $(150, 200)$ used in most of them.

Stochastic Weight Averaging

Studying more closely loss surfaces for deep neural networks, Izmailov *et al.* proposed the Stochastic Weight Averaging (SWA) method for creating models with better generalization, based on the parameters (weights) of the intermediary models obtained during a training with the standard Stochastic Gradient Descent algorithm, with a cyclical or constant learning rate. This method consists in averaging the weights of the model in pre-selected points during the training procedure. This is justifiable because they found out that the learning algorithm will surround generalization optima points, never actually reaching them. They also uncovered that optima points on the training loss surface are usually shifted with respect to the test loss surface. Other properties are described and further justify this strategy. For cyclical learning rates, they chose points so that the learning rate reaches is minimum value in the cycle; for constant learning rate, a point is chosen in each epoch [IPG+18].

Park used this strategy to improve the model's ability to learn relevant feature vectors. As shown in this section, a scheduler is used to configure the learning rate in Park's model. However, since no change is done after the last milestone, it is justifiable to apply SWA. In the selection of averaging points, the highest validation average score of the last 20 epochs is considered, and when this highest score is reached, the last 10 epochs are averaged. Moreover, Park uses a *moving* average, which assigns a decreasing weight in chronological order given by rate $\alpha = 1/(1+i)$, where i is the ordinal number of the model being averaged, *i.e.* less weight is assigned to the models closer to the last point. Departing from the standard approach, which is to average during training, here the model points are combined after the training ends, in a post-training step.

5.4 Inference Phase

Unlike other competitors who used classification, Park decided to base his solution's inference on surface areas on a hypersphere. As shown in Section 5.3, the network's final output (considering a single image as input) is a feature vector with 512 entries. This 512-dimensional space is the one used to characterize the whales, having the cosine similarity of normalized features as comparison metric.

In order to create a template for each class, the trained neural network is used to infer the feature factors for the training dataset images, considering only the identifiable whales, *i.e.* excluding the *new whale* class.

However, these features produced by the network are directly related to a single image, not to the whale's identity. Therefore, another step has to follow feature inference, in order to transform or aggregate multi-representations learned for each identity to create embedding features. Park based his approach on the ArcFace proposal [DGXZ19], which is to compute the feature center and use it as the embedding features of a class with multiple examples. Although this could be applied directly, Park decided to refine even further the strategy, by performing the centering process only with the closest features with respect to the center. When less than four images are available for a given identity, the feature center (mean) is defined simply as the embedding feature. Otherwise, an iterative process with two-step reduction is used. First, the center is computed using all images' features, then a fourth of the features are discarded (the farthest from the center). This process is repeated once again if more than three representations are still available. Finally, the center is computed for the remaining features. Once the center is determined, a normalization is done to create a correspondence between feature vectors and the hypersphere surface.

In addition, features are computed both using the fluke alignment (yet conditioned to a 50% chance of not being aligned) and not using it. Besides, because of the flipping trick (see Section 5.2), another set of feature embeddings is created considering only flipped images, also varying the alignment.

Then, the test image feature inferences are done, also having the original and flipped identities' embedding created separately with both changing the alignment to increase the number of feature vectors. After this process, a similarity matrix is created using the cosine similarity, described in Section 5.4.1, for both flipped and non-flipped images. These similarities are then combined by averaging the results of the correspondent non-flipped and flipped identities. This resulting matrix is then used to decide which classes are more likely to correspond to each test image. This final step is described in Section 5.5, along with the assembling process of different trained models.

Model	Scheduler Milestones	Learning Rate	Class Grouping
1st	$(250, 350)$	5×10^{-4}	-
2nd	$(150, 200)$	5×10^{-4}	-
3rd	$(150, 200)$	2.5×10^{-4}	✓

Table 5.2. Changes in configurations used by Park to create different trained models and generate the final identification. For more information about scheduler milestones, see Section 5.3.3. All other parameters mentioned throughout the chapter are kept unchanged.

5.4.1 Cosine Similarity

Cosine similarity is a metric that uses the information of the angle between vectors u and v, through its cosine, to measure distance. Since the vectors are normalized beforehand, a simple way to obtain the cosine of the smallest angle formed by them is by the inner product:

$$\langle u, v\rangle = v^T u = \|v\|\|u\| \cos\theta = \cos\theta. \tag{5.1}$$

When $\cos\theta = 1$, it follows that $\theta = 0$, *i.e.* $u = v$, since the vectors are normalized. On the other hand, if $\cos\theta = -1$, then $\theta = \pi$ and $u = -v$. In other words, when the cosine is close to 1, both vectors are pointing to a close position in the hypersphere, and they are pointing to opposite directions when it is close to -1. Therefore, one can use this information to identify whether two vectors point to a same position within a margin, which is analogous to have the classes spread on the surface of the hypersphere.

5.5 Ensemble Method

To create the submission classification, an ensemble method was conceived to combine the knowledge obtained by three variants of the model. These variations consisted in changes in the training scheduler configuration, learning rate, and label corrections by class grouping, described in Section 5.2.2. Table 5.2 summarizes the differences among these three configurations.

Once the cosine similarity matrix is computed for all three models, following the specified in Section 5.4, the assembling process in done by computing the average of the cosine similarity matrices. This combined result is then used to selected the five most likely whale identities for a given image. Because the matrix does not include the *new whale* class (since this is actually not a unique whale which can be discriminated by a single feature embedding), a fixed threshold $\alpha = 0.385$ is considered to classify a whale as new, *i.e.* when the similarity of the next most similar class is less than α, the *new whale* class is used as the current guess. This threshold value was chosen, as explained by Park in his solution overview post [Par19a], to yield *new whale* in about 27.6% of the top-1 predictions.

Model	Leaderboard score	
	Public	Private
Park's	0.97419	0.97113
Ours	0.96984	0.96889

Table 5.3. Official results on public and private leaderboards on Kaggle Platform for Park's final submission and our submission.

Finally, top-1 class corrections and conversion of grouped classes are done as a post-processing step, as specified in Sections 5.2.3 and 5.2.2.

5.6 Reproducing Solution Training

Park structured his solution to deal with two tasks: generating landmarks and identifying the whales by their tails. Thus, we have configuration files to hande each task. However, his landmark predictions are available in the solution repository on GitHub [Par19b]. Park also made his trained model available for download. Since our purposes include checking the reproducibility of the solutions, we retrained the model used to extract features from the images. Therefore, three models were trained using the configuration files available in the repository. All parameters shown throughout this chapter are specified either in these configuration files or directly in the source code.

To adapt the training to our resources, we increased the batch sizes from 32 to 96 samples. The number of epochs is not directly given by Park on his solution overview post, but it is shown indirectly to be 300 when he specifies the expected training time. Despite that, we decided to train it for 500 epochs, following the specification in the configuration files.

After training each model following the configuration shown in Section 5.5, we combined the network model points using the SWA strategy, with code found in the repository. After that, we generated the similarity matrices for all models, combined them, and generated the final classification result, also using source code available to performing this task. The final result obtained by this process was then submitted to the *Kaggle* Platform. The results are shown in Table 5.3.

5.7 Conclusion

Despite not approaching the task with a classification network, Park was able to create a high confidence solution. By using simple data transformations and a more sophisticated alignment technique, he created variations to prevent overfitting and improve generalization. He also exploited a state-of-the-art neural network to compose flukes' feature predictions: DenseNet. This network exploits even further the idea of creating short connections among layers by creating fully connected blocks.

In addition, the ArcFace strategy is used to guarantee inter-class discrimination power for this deep network. As a result, 512-D feature vectors can be created for each identity, and then used to identify unseen whales by their tails. An ensemble is also used to improve the predictions, as well as label corrections in a final processing step. Finally, our training reproduction led to close results in both public and private leaderboards, displaying absolute differences around 0.004 and 0.002, respectively, between our scores and the ones obtained by the author.

Chapter 6

Acknowledgments

This work was supported by the São Paulo Research Foundation (FAPESP) under grant no. 2019/11386-3. We also acknowledge the availability of computational resources described in Section 1.2, acquired through FAPESP support under grant no. 2018/00031-7, and maintained by the Institute of Computing/UNICAMP.

We also thank the Kaggle platform for keeping the competition available even after its deadline, in the *late submission* format, and the Kaggle community for the sharing of solutions and comments.

Opinions, hypothesis and conclusions, or recommendations expressed in this material are responsibility of the author(s), and do not necessarily reflect FAPESP's point of view.

Bibliography

[AB07] Tuncer C Aysal and Kenneth E Barner. Rayleigh-maximum-likelihood filtering for speckle reduction of ultrasound images. *IEEE Transactions on Medical Imaging*, 26(5):712–727, 2007.

[Bad19] Malek Badreddine. Bad quality images. `https://www.kaggle.com/c/humpback-whale-identification/discussion/73882`, 2019. Accessed: 2019-12-16.

[Cad17] Remi Cadene. Pretrained models for PyTorch. `https://github.com/Cadene/pretrained-models.pytorch`, 2017. Accessed: 2020-05-17.

[CF97] Christine Connolly and T Fleiss. A study of efficiency and accuracy in the transformation from RGB to CIELAB color space. *IEEE Transactions on Image Processing*, 6(7):1046–1048, 1997.

[CH67] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[Che19] Heng CherKeng. Here are the tricks. `https://www.kaggle.com/c/humpback-whale-identification/discussion/79384`, 2019. Accessed: 2020-01-20.

[CWP$^+$18] Yilun Chen, Zhicheng Wang, Yuxiang Peng, Zhiqiang Zhang, Gang Yu, and Jian Sun. Cascaded pyramid network for multi-person pose estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7103–7112, 2018.

[DGXZ19] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4690–4699, 2019.

[DK16] Samuel Dodge and Lina Karam. Understanding how image quality affects deep neural networks. In *IEEE International conference on quality of multimedia experience (QoMEX)*, pages 1–6. IEEE, 2016.

[Doc20] Docker Inc. Docker: Empowering app development for developers. `https://www.docker.com/`, 2020. Accessed: 2020-01-31.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GKXS18] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *arXiv preprint arXiv:1810.13243*, 2018.

[GZ17] Yandong Guo and Lei Zhang. One-shot face recognition by promoting underrepresented classes. *arXiv preprint arXiv:1707.05574*, 2017.

[HBL17] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.

[HSS18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7132–7141, 2018.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[IPG$^+$18] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[Joh19] Paul Johnson. 1000 Hand-Annotated Humpback Whale Fluke Keypoints. `https://www.kaggle.com/c/humpback-whale-identification/discussion/78699`, 2019. Accessed: 2020-05-04.

[Kag18] Kaggle. Humpback Whale Identification Challenge: Can you identify a whale by the picture of its fluke? `https://www.kaggle.com/c/whale-categorization-playground`, 2018. Accessed: 2019-12-24.

[Kag19a] Kaggle. How to use Kaggle: Competitions. `https://www.kaggle.com/docs/competitions`, 2019. Accessed: 2019-12-18.

[Kag19b] Kaggle. Humpback Whale Identification: Can you identify a whale by its tail? `https://www.kaggle.com/c/humpback-whale-identification/overview/description`, 2019. Accessed: 2019-12-18.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KC17] Michał Koziarski and Bogusław Cyganek. Image recognition with deep neural networks in presence of noise–dealing with and taking advantage of distortions. *Integrated Computer-Aided Engineering*, 24(4):337–349, 2017.

[LC19] Fredrik Lundh and Contributors. Pillow documentation. `https://pillow.readthedocs.io/en/stable`, 2019. Accessed: 2019-12-17.

[LGG+17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2980–2988, 2017.

[Mil95] George A Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[Mok19] Oleksandr Mokin. List of duplicate whale ids. `https://www.kaggle.com/c/humpback-whale-identification/discussion/81885`, 2019. Accessed: 2020-05-08.

[Num19] NumPy. NumPy documentation. `https://numpy.org`, 2019. Accessed: 2019-12-17.

[Osm19] Radek Osmulski. Fluke detection using fastai. `https://www.kaggle.com/c/humpback-whale-identification/discussion/76281`, 2019. Accessed: 2020-05-04.

[PAP+86] Stephen M Pizer, John D Austin, John R Perry, Hal D Safrit, and John B Zimmerman. Adaptive histogram equalization for automatic contrast enhancement of medical images. In *Application of Optical Instrumentation in Medicine XIV and Picture Archiving and Communication Systems*, volume 626, pages 242–250. International Society for Optics and Photonics, 1986.

[Par19a] Jinmo "Pudae" Park. 3rd place solution with code: ArcFace. `https://www.kaggle.com/c/humpback-whale-identification/discussion/82484`, 2019. Accessed: 2020-04-29.

[Par19b] Jinmo "Pudae" Park. Code for 3rd place solution in Kaggle Humpback Whale Identification Challenge. `https://github.com/pudae/kaggle-humpback`, 2019. Accessed: 2020-04-29.

[PY09] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2009.

[Qia19] Jian “Earhian” Qiao. 1st place solution. `https://www.kaggle.com/c/humpback-whale-identification/discussion/82366`, 2019. Accessed: 2019-12-18.

[QLTW19] Jian Qiao, Peiyuan Liao, Thomas Tilli, and Yiheng Wang. Kaggle humpback whale identification challenge 1st place code. `https://github.com/earhian/Humpback-Whale-Identification-1st-`, 2019. Accessed: 2019-12-18.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal on Computer Vision*, 115(3):211–252, 2015.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241, Cham, 2015. Springer International Publishing.

[She19a] Tao Shen. 2nd place code: end to end whale identification model. `https://www.kaggle.com/c/humpback-whale-identification/discussion/82366`, 2019. Accessed: 2020-03-27.

[She19b] Tao Shen. Humpback Whale Identification Challenge 2019: 2nd place solution. `https://github.com/SeuTao/Humpback-Whale-Identification-Challenge-2019_2nd_palce_solution`, 2019. Accessed: 2020-02-21.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, 2015.

[SZY+18] Yifan Sun, Liang Zheng, Yi Yang, Qi Tian, and Shengjin Wang. Beyond part models: Person retrieval with refined part pooling (and a strong convolutional baseline). In *European Conference on Computer Vision (ECCV)*, pages 480–496, 2018.

[Wan19] Yiheng “Venn” Wang. Open source all leaks (135 samples) I detected. `https://www.kaggle.com/c/humpback-whale-identification/discussion/80086`, 2019. Accessed: 2020-05-08.

[WCLL18] Feng Wang, Jian Cheng, Weiyang Liu, and Haijun Liu. Additive Margin Softmax for Face Verification. *IEEE Signal Processing Letters*, 25(7):926–930, 2018.

[WS09] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *The Journal of Machine Learning Research*, 10(Feb):207–244, 2009.

[WWZ+18] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5265–5274, 2018.

[WXCY17] Feng Wang, Xiang Xiang, Jian Cheng, and Alan Loddon Yuille. Normface: L2 hypersphere embedding for face verification. In *ACM International Conference on Multimedia*, pages 1041–1049, 2017.

[XGD+17] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1492–1500, 2017.

[Yan17] Ming Yang. Deepo: set up Deep Learning environment in a single command line. `https://github.com/ufoym/deepo`, 2017. Accessed: 2020-02-13.

[ZZK+17] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random Erasing Data Augmentation. *arXiv preprint arXiv:1708.04896*, 2017.

Appendices

Appendix A

Extra Bad Quality Images

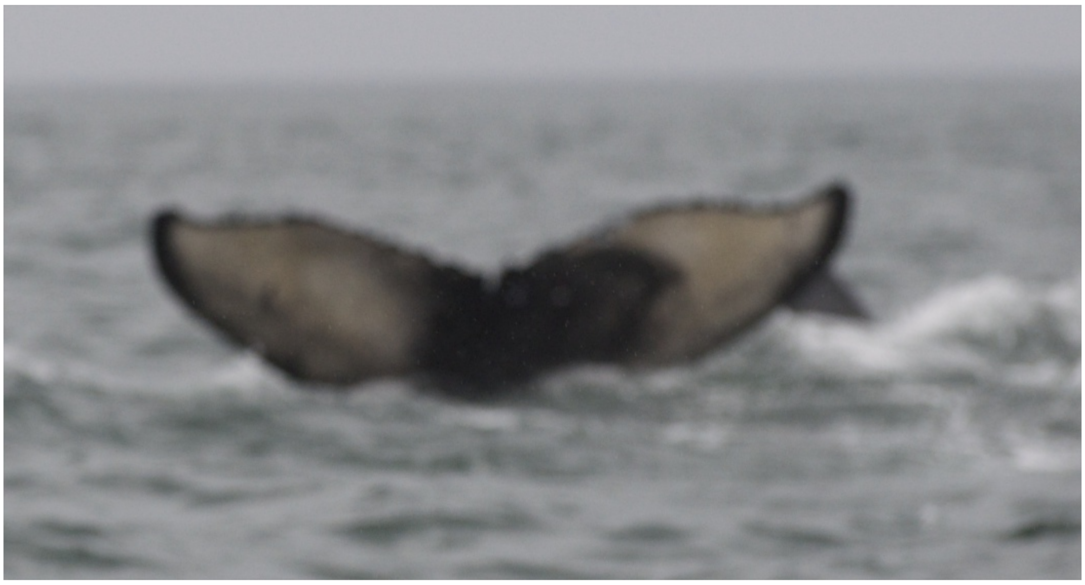


Figure A.1. Example of a very blurry image.

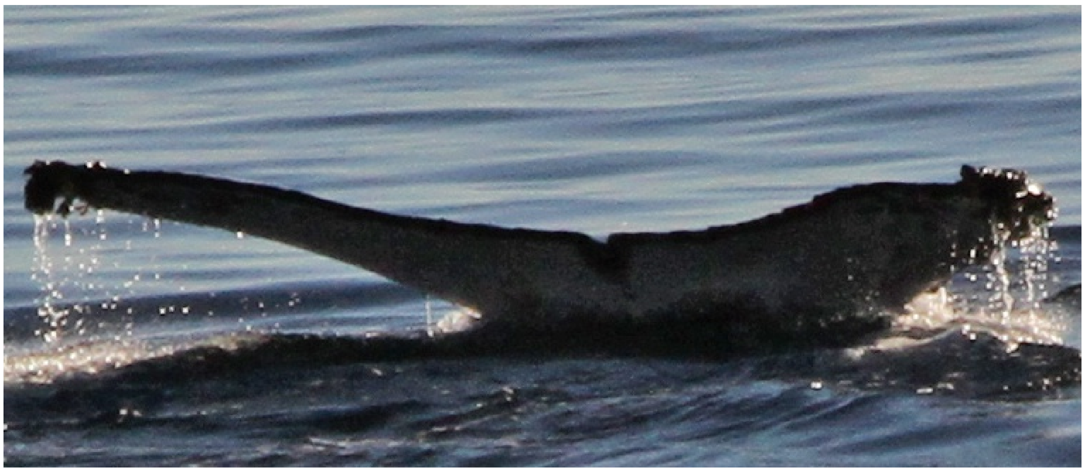


Figure A.2. Poor picture angle.

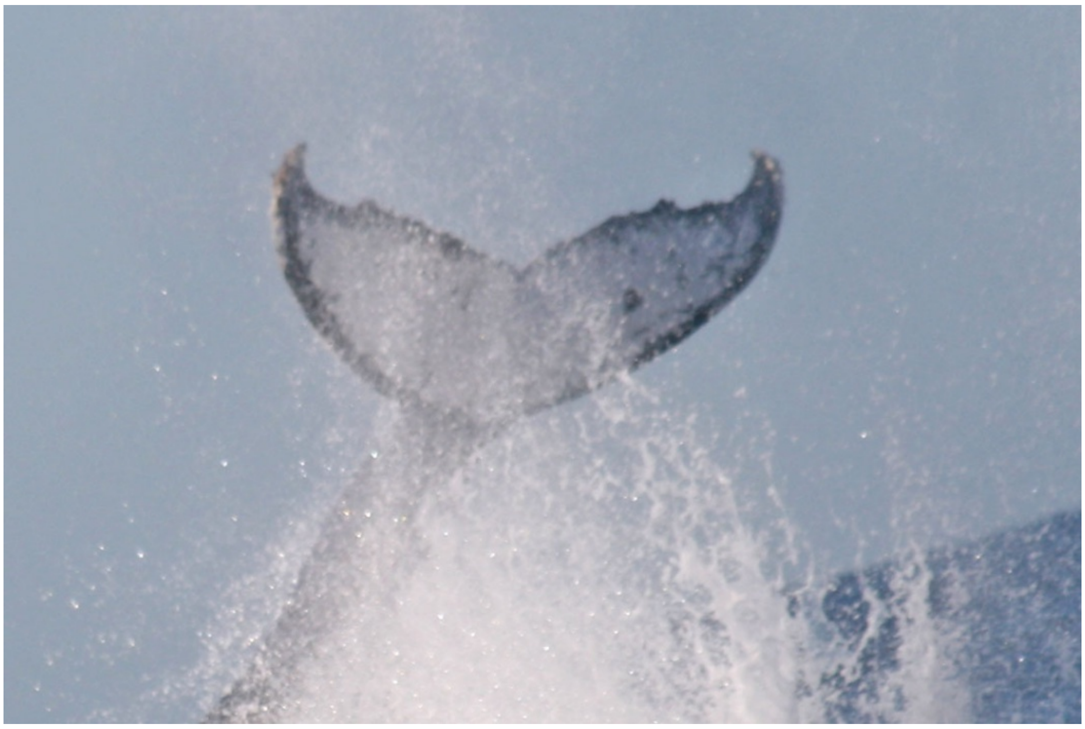


Figure A.3. Whale's tail occluded by water spray.

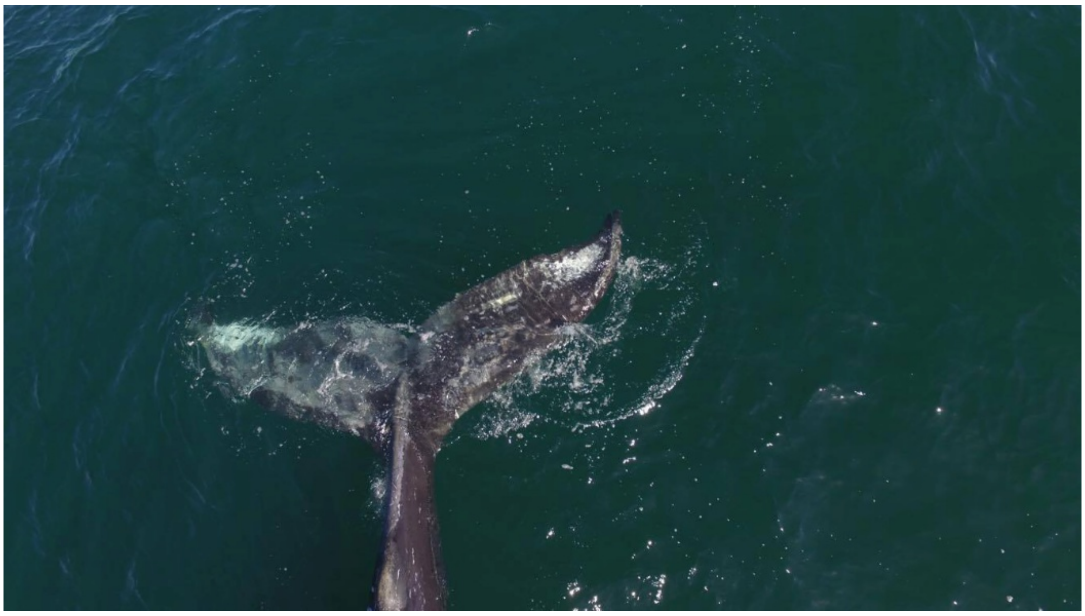


Figure A.4. Front side of the fluke instead of the back, and partially occluded by water.

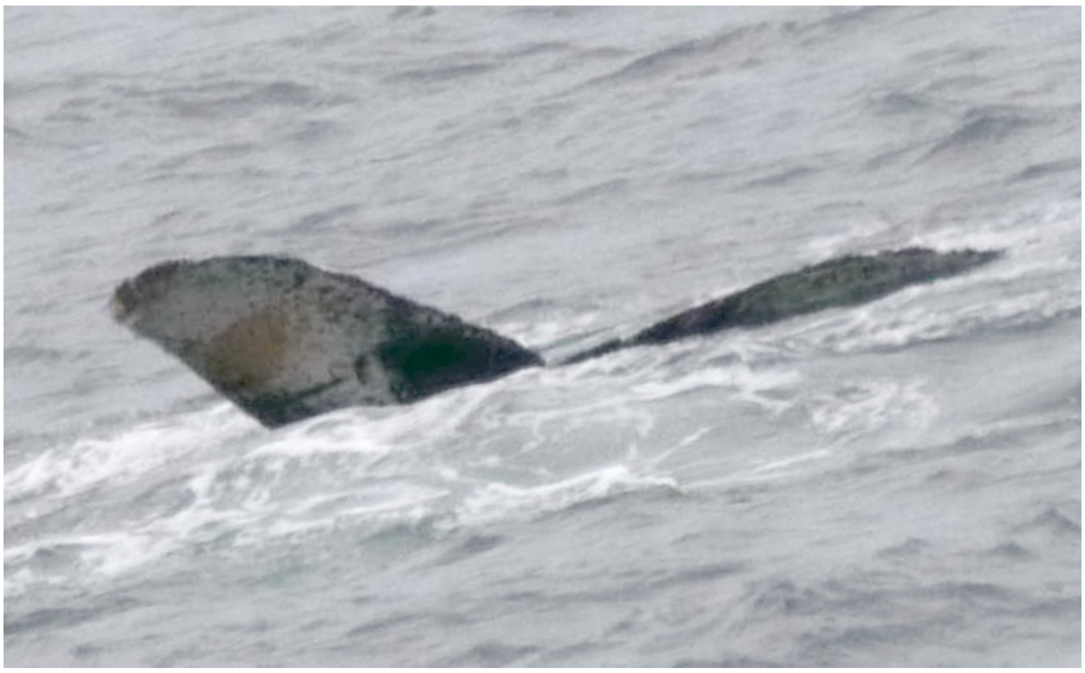


Figure A.5. Partial occlusion of the tail, due to water immersion.

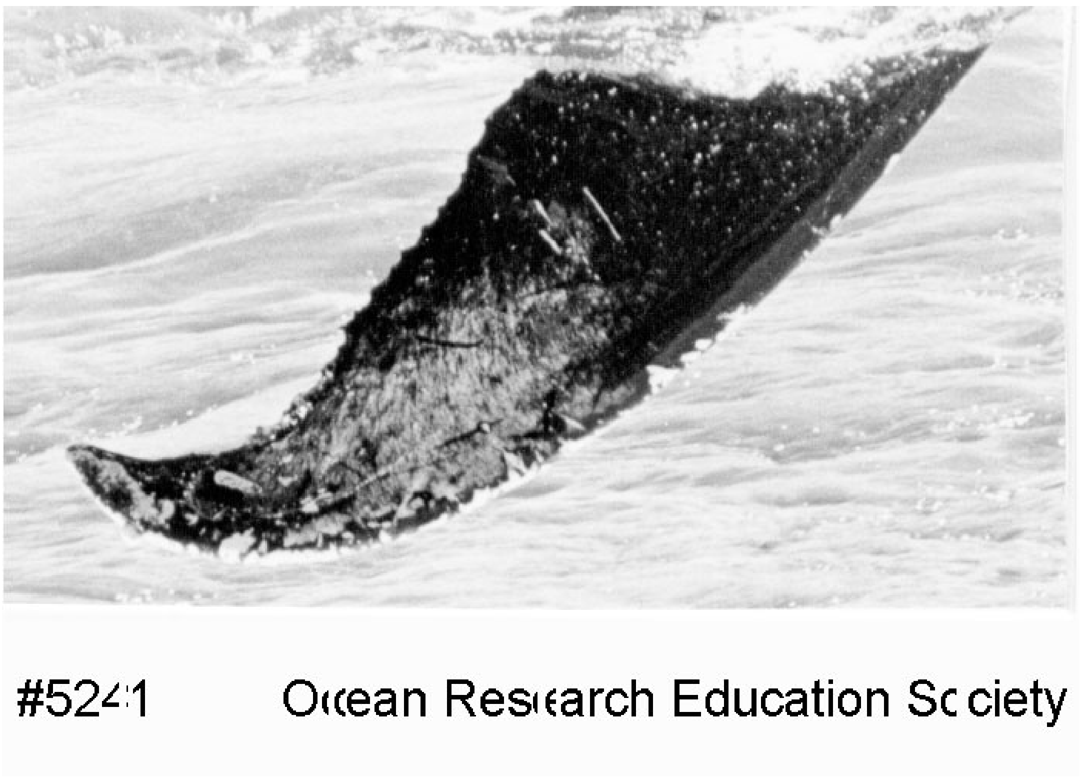


Figure A.6. Partially occluded and with text in the image.