# Open Kaggle: Humpback Whale Identification Challenge

## Abstract

In this paper, we present our insights and solution to the Kaggle Humpback Whale Identification challenge, where individual whales must be identified by a picture of their fluke. The data set is the Happy Whales database of over 25,000 images, gathered from research institutions and public contributors. Since this data set has only a couple of images per class (per whale), the task at hand falls into the category of Few-Shot Learning problems. The performances of two different Machine Learning algorithms have been compared: a regular Convolutional Neural Network and a so called Siamese Network Architecture, which is a CNN variant tailored to Few-Shot Learning tasks. In order to improve model performance, the images have been preprocessed by automatically cropping the areas of interest. Ultimately we found that the CNN slightly outperformed the Siamese Network, however we see more potential in the latter approach.

## 1. Introduction

Whale populations have been suffering from industrial fishing and warming ocean temperatures. As an effort to avoid extinction, Scientists have been involved in conserving whale species by monitoring their ocean activity through photo surveillance systems. While they use this image data to identify whales and precisely log any activities, the process has been done manually by a few scientists for the last decades.

In order to improve the monitoring of whales, leave less data untapped and make the process scalable to bigger amounts of data, an automated image identification system is needed.

This work presents and compares two algorithms, that take whale images as an input, extract features automatically and output the classification of the whale images based on CNNs and their adaptions. We use the Happy Whales Dataset on kaggle, which includes images of humpback flukes. The challenge lies in dealing with a skewed class distribution. While a few classes have many samples, the majority of the classes contain less than 3 samples, making the problem subject to Few-Shot Learning. Our main approach aims to cope with Few-Shot learning by not computing the class probabilities directly, but by first determining similarities between images based on their lower dimensional embeddings and then making the classification.

The outline of the paper is as follows: Section 2 gives an overview of previously published solutions on whale identification methods and highlights their limitations. Section 3 analyzes the characteristics and challenges given by the underlying data set. The preprocessing pipeline of the images is explained in section 4, while section 5 gives insight about the theory and architecture of the two algorithms used for whale identification. In section 6, we quantify the performance of both algorithms. In conclusion, we provide a summary of the winner's solution of this challenge and sum up main takeaways.

## 2. Related Work

Although done manually so far, computer-based approaches to identify humpback whales already exist: Ranguelova et al. introduced an identification framework based on light and dark pigmentation patches on the whale flukes. Kniest & Burns propose a software to manually query images and compare them to an existing catalogue of humpback images. This comparison to the catalogue is made by using fixed features, such as black and white patches, certain shapes and edges, etc. as well as user defined features which describe spots, areas, damages or other image features. Both approaches have in common, that features are hand crafted and are not learned by an algorithm. Another similarity in these two approaches is, that users have to actively use a GUI to specify and mark areas of the query images, making the identification task not an automatic but a semi-automatic one. Assessing both the hand crafting of features as well as the necessity to specify key regions through a GUI, it can be argued that there is room for

. Correspondence to: Max Botler <max.botler@campus.tu-berlin.de>, Alexandru Bundea <alexandru.bundea@campus.tu-berlin.de>, Clemens Peters <clemens.peters@campus.tu-berlin.de>, Marius Schidlak <marius.schidlack@campus.tu-berlin.de>.

further improvement through the automatic extraction of features as well as an increased scalability by automatically identifying the whales.

The same task but for a different domain was given by the *Right Whale Recognition* competition on *kaggle*. Aim was to also classify the individual whale, given an image with an aerial shot from a right whale. The winner of this competition, as well as other leading competitors, used a pipeline of regular ConvNets to first extract the region of interest - which was a characteristic pattern on their heads - and to then classify based on those extracted regions (for the winning framework refer to (deepsense.ai, 2016)). The class distribution of the underlying data set of this competition was less imbalanced and the ratio of classes with only a few images compared to the total number of classes was rather low. Hence, the proposed algorithms are not catering to the problematic of Few-Shot Learning.

A framework catering towards an automatic feature extraction, scalability as well as to the problematic of Few-Shot Learning is presented in Schroff et al.. We are transferring the idea of clustering and recognizing faces to the clustering and recognition of unique whale flukes.

## 3. Dataset

### 3.1. Exploration

The training set consists of 9850 images while the test set consists of 15610 images.

In total, there are 4251 classes inside the dataset, where each class represents a whale. Although it is important to discriminate between the class ”new whale” and the rest of the classes. ”New whale” is the class for an image which hasn’t been labelled and serves as a ”fallback”-class.

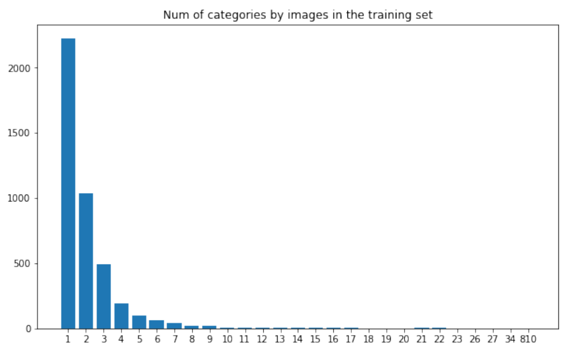
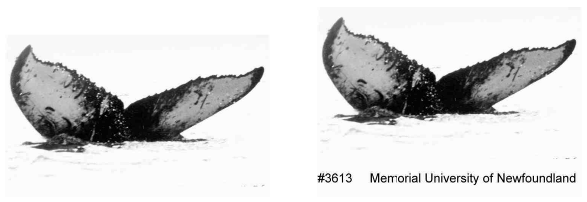


*Figure 1.* Distribution of training samples for all classes (Toumbourou, 2018)

Figure 1 depicts how many classes only have a certain amount of samples. For example: over 2000 classes only have 1 training sample while close to 500 classes have 3 training samples. Interestingly, 810 training samples belong to the ”new whale” class.
But not only the training set consisted of ”new whales”. This has been verified by uploading a submission to the kaggle challenge by setting ”new whale” as the first and only choice: the observed precision resulted at approximately 32,5%.
Furthermore, duplicate pictures have been found inside the test set. Moreover, many images inside the training samples are altered versions of other samples. For example: we found ”pseudo duplicates”, which are duplicates by being cropped in another way. Figure 2 depicts an example of a duplicate pair. The image 2a has been centered with the fluke in the center, while 2b has a subtitle below and displays the fluke on the top of the image.



(a) image 5705158b with label w_8158b1f

(b) image c9144fba with label w_8158b1f

*Figure 2.* Pseudo Duplicates: image b is a duplicate of image a, while the fluke is displayed on the top part of the image and a caption has been added

Figure 3 presents an overview of all issue groups inside the training set.

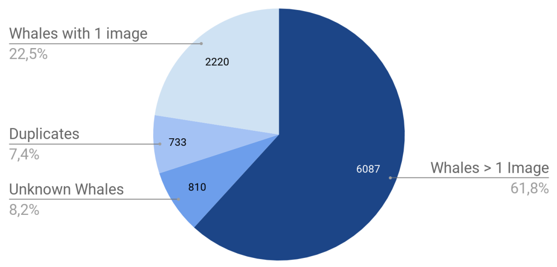


*Figure 3.* Issue groups of the training set

Regarding the quality of the images, several differences among the images can be found. For example, the resolution of the images is not uniform. Figure 4 shows the distribution of training samples per pixel resolution. The rectangular shape of the images suggests, that most images have been cropped to center the fluke better.

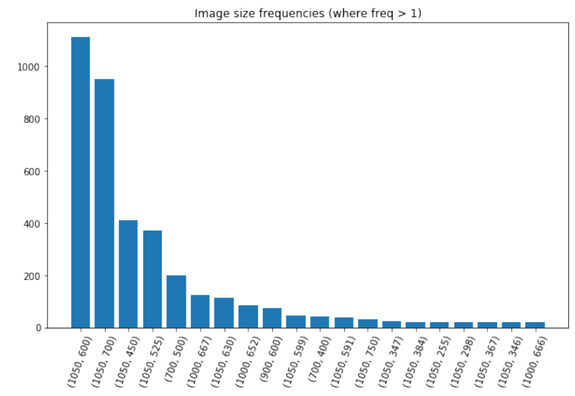
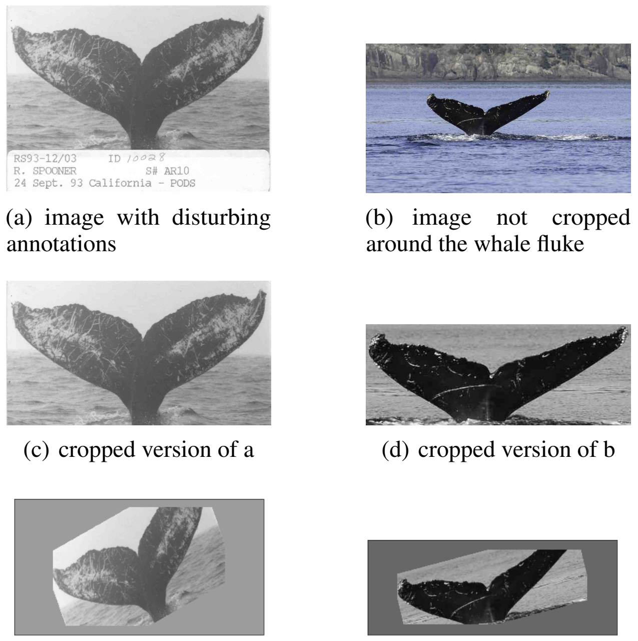


*Figure 4.* Distribution of picture pixel size inside the training set (Toumbourou, 2018)

Another problem inside the training set has been detected by studying particular images only display a part of a fluke. Similarly, a certain amount of training samples do not even display a fluke, but e.g. an eye of a whale or display a rear side-view of a whale from far away.
All aforementioned issues have to led to the idea of using the knowledge of the duplicates and the class ”new whale” to test how much accuracy can be achieved with any machine learning. The test has been done via another submission to the kaggle challenge with the following class propositions: if the image is exactly a duplicate image which has been inside the training set, its label got selected as the first proposition, additionally ”new whale” is the second proposition. If there is no duplicate, ”new whale” is the first proposition. This approach has resulted in a 40,8% accuracy. This means, that without the use of any machine learning algorithms, the actual baseline of this competition is at 40,8% accuracy.
Since approximately, 80% of classes have three or less training images, the problem falls in the category of ”few-shot-learning”.

### 3.2. Training and Validation Split

As mentionned before, the training set consists of 9850 images, which needs to be cleaned from the duplicate, the 810 samples belonging to ”new_whale”. After this data cleaning, approximately 6000 training samples remained.
Although the data cleaning already has shrunk the training data significantly, the training set also needs to be seperated into a validation set. Attention has to be payed for choosing the validation set for this classification task: for the training of few-shot-learning methods, we need to give each class at least two samples, one in the validation set and at least one in the training set. Out of the 2050 classes with two or more training images, we placed one sample in the validation set, while the training set now only consists of 4155 samples with three augmentations each, totalling 16620 training samples.



(a) image with disturbing annotations

(b) image not cropped around the whale fluke

(c) cropped version of a

(d) cropped version of b

(e) sample preprocessed version of a

(f) sample preprocessed version of b

*Figure 5.* Preprocessing: Before and after processing two exemplary images

## 4. Preprocessing

After experimenting with different approaches, we found these Preprocessing steps to contribute most to the performance of our final model:

- Convert to Gray-Scale,
- Retrieve the bounding boxes of the flukes,
- Augmentations (only during training of the classifier) and
- Resize Image

### 4.1. Convert to Gray-Scale

In this step, we converted the images to black and white. As Piotte elaborates, the models perform poorly between comparing image pairs of colored and gray-scale images, while the accuracy does not differ much when only using gray-scale images.

### 4.2. Bounding Box Model

Most of the images are already cropped around the whale fluke. Yet, there are still images in both train and test set that contain noise that do not provide any useful information for feature extraction. Such noise was mainly due to unnecessary background on one side. On the other side, image annotations occurred in about 10% of all images. Figure 5 (a) and (b) show exemplary images with the respective noise.

In order to target this noise and only provide relevant image parts for the classification model, we trained a bounding box model to predict the 4 coordinates characterizing the position of the boxes - more precise the min and max values of the vertical and horizontal axis of the image - based on a VGG-Net like architecture. The idea of this approach, including its implementation, was obtained from Piotte.

The bounding box model architecture can be divided into three main parts. The first part consists of two convolutional layers. Different to the original VGG-Net, the kernel sizes were chosen to be larger. The convolutional layers were followed by applying batch normalization and dropout.
The second part consists of in total 5 subsequent blocks. Each block contains three convolutional layers, one batch norm and one dropout layer. The first convolutional layer of each block has a kernel size of 2x2, a stride of 2 and is intended to replace regular max pooling in order to loose less information about the location of features.
The third part aims to address horizontal features for the horizontal coordinates and vice versa for the vertical. This is achieved by using different max pooling sizes. Intuitively, max pooling should only merge horizontal features together and not take the verticals into account. The same applies to the opposite calculation for the vertical features. Hence, the third part of this architecture utilizes two separate streamlines, where each streamline uses a custom kernel for max pooling for each horizontal and vertical features (see table 1 after layer 'dropout6').

For the actual training of the model, a manually labeled data set created by Piotte was used and contained 1200 images from the original data set, where 1000 images were used for training and 200 for validation. The whole bounding box training pipeline contains the following steps:

**Preprocessing of Images** After converting the images to black and white, random affine transformations were applied and included rotation, sheer, zoom and shift. Note at this point, that the same transformations have to be applied to the ground truth labels. That followed, the image was resized to 128x128 pixels and normalized to 0 mean and variance 1.

**Train And Select Best Bounding Box Model** The model explained above was trained using Adam Optimizer with a learning rate of 0.032 and Mean Squared Error as loss function. The model was trained for either 50 epochs or after the early stopping criteria - delta in validation loss smaller than 0.1 for the last 9 epochs - was fulfilled. Therefore, the learning rate was reduced by the factor 0.25 if the validation loss did not decrease more than 0.1 for the last 3 epochs. We repeated training 3 times and selected the model that achieved the lowest validation loss.

**Variance Normalization** As pointed out by Li et al., using Dropout shifts the variance of a neuron activation when transferring from train to inference mode. Since Batch Normalization on the other side uses the variance accumulated throughout the training phase, the performance can suffer due to the inconsistency of the variance of the neuron activation. In order to avoid this problem, the model previously selected was trained for another 1 epoch while deactivating all dropout layers and using a rather small learning rate of 0.002. This yielded in a reduction of the total loss on the validation set of 12% comparing the loss values before and after Variance Normalization.

For inference, the bounding box model uses the same preprocessing steps except applying random affine transformations. After computing the bounding box coordinates for the image with shape 128x128, the bounding box coordinates get transformed back to the original image size.

See figure 1 (c) and (d) for the result of the cropping through bounding box extraction.

| layer | size-out | kernel | params |
|---|---|---|---|
| input | 128x128x1 | - | 0 |
| conv1 | 128x128x64 | 9x9, 64 | 5,248 |
| conv2 | 128x128x64 | 3x3, 64 | 36,928 |
| bnorm1 | 128x128x64 | - | 256 |
| dropout1 | 128x128x64 | - | 0 |
| ... | ... | ... | ... |
| conv15 | 4x4x64 | 2x2, 64, stride 2 | 16,448 |
| conv16 | 4x4x64 | 3x3, 64 | 36,928 |
| conv17 | 4x4x64 | 3x3, 64 | 36,928 |
| bnorm6 | 4x4x64 | - | 256 |
| dropout6 | 4x4x64 | - | 0 |
| pool1a | 4x1x64 | stride 4x1 | 0 |
| pool1b | 1x4x64 | stride 1x4 | 0 |
| flatten1a | 256 | - | 0 |
| flatten2b | 256 | - | 0 |
| dropout7a | 256 | - | 0 |
| dropout7b | 256 | - | 0 |
| dense1a | 16 | - | 4,112 |
| dense1b | 16 | - | 4,112 |
| concat1 | 32 | - | 0 |
| dropout8 | 32 | - | 0 |
| dense2 | 4 | - | 132 |
| total | | | 503,588 |

*Table 1.* Architecture of the Bounding Box Model

### 4.3. Augmentations

For increasing the robustness of the classifier to certain invariances, random augmentations as shown in figure 1 (e) and (f) were applied to the train images. Those augmentations consisted of random zoom and rotation.

### 4.4. Resize Image

Since common classifiers, such as ConvNets, need a constant input size, we chose to resize all images to 320x205 pixels. We tried to balance the compromise between using as much pixels as possible to retain all information necessary while keeping the amount of pixels low in order decrease computation efforts. Therefore, the aspect ratio is based on a subjective estimation of the aspect ratio of a whale fluke. Extracting and using the median aspect ratio of all cropped fluke images did not yield in better validation and test results.

## 5. Method

### 5.1. Regular Convolutional Neural Net

The CNN architecture became a very popular tool in image recognition tasks in recent years. It was successfully applied in many problem settings recently and has proven to produce very good results (Szegedy et al., 2016). For these reasons, it serves as the baseline approach in this paper. Comparing other, more complex architectures to the standard CNN allows to choose an appropriate model for the given task. In order to work computationally efficent, avoid high costs and long training times, we use a pretrained network as a basis. The InceptionV3 model (Szegedy et al., 2016), with weights pre-trained on ImageNet is already included in Keras. The fully-connected layer at the top of the network was not included. Instead, one additional dense layer with softmax activation function was added to the InceptionV3 model. Additionally we used dropout (with probability $0.5$) to prevent overfitting on our relatively small dataset. As loss function we chose categorical crossentropy. All weights of the model were frozen and only the weights of the last layer were adapted during the training process. This allows for a fast and efficient training, reusing the basic pretrained filters.

Speed up in training can be achieved by precomputing the feature vectors obtained from feeding the input through the InceptionV3 net without the top layers. Using this procedure, only training the last dense layer (while applying dropout) has to be done without computing the feature vectors for each training pass.

### 5.2. Siamese Network

As an alternative to the regular CNN, we use a so called Siamese Network architecture. The architecture is a variation of Convolutional Neural Networks that is tailored to Few-Shot Learning problems. It has been used successfully for tasks similar to the one at hand, such as the identification of individual faces by a small amount of training samples for each face. Furthermore the architecture allows for an arbitrary number of classes. One such architecture was employed in the FaceNet-System Schroff et al. and led to an 99.63% accuracy on the Labeled Faces in the Wild (LFW) dataset. The architecture described in the following sections is highly inspired by FaceNet.

Instead of learning a probability distribution and directly predicting the probability of an image belonging to a particular class, the Siamese Network learns a function that measures similarity between two images. Namely, how similar a given image of a whale fluke is to the images from the training data. The images most similar to the predicted image will then yield the predicted classes.

In order to achieve just that, the model is divided into two different parts, which we will call the Top-Model and the Bottom-Model in the following.

The Top-Model learns to produce and embeddings of the data, i.e. it projects the images into a euclidean space of arbitrary dimension. These embeddings are learned, such that images of the same class have a small euclidean distance to one another, while images of different classes are far apart.

The Bottom-Model then utilizes this similarity measure to to predict the classes of images.

#### 5.2.1. Technical Introduction

In the following section we will give an overview of the learning techniques associated with Siamese Networks.

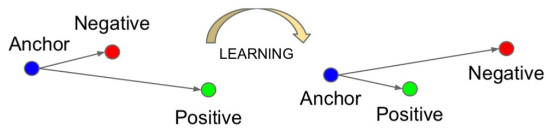


*Figure 6.* Goal of triplet learning (Schroff et al., 2015)

**Triplet-Loss** For learning the Top-Model, a special loss function called Triplet-Loss is introduced (Schroff et al., 2015). As the name indicates, the Triplet-Loss is defined on triplets of data points. Each triplet consists of one anchor point, a corresponding positive example with the same class as the anchor and one negative example which has a different class than the anchor. As hinted before, the goal of the learning process is that for all possible triplets, the positive example is closer to the anchor than the negative (cf. figure 6).

Put formally, each triplet $(a_i, p_i, n_i)$ should satisfy the condition

$$\|f(a_i) - f(p_i)\|_2^2 + \alpha \leq \|f(a_i) - f(n_i)\|_2^2 \qquad (1)$$

Where f(x) is the model that computes the embedding and $\alpha$ is a margin (i.e. a minimum distance) that is enforced between the positive and the negative example. This margin is also necessary because setting $f(x) = 0$ would trivially fulfill the condition.

If a triplet already satisfies the condition, we thus consider it to be embedded correctly and set the corresponding loss to zero. Putting it all together we obtain the Triplet-Loss function

$$L = \sum_{i=1}^{N} max(0, \|f(a_i) - f(p_i)\|_2^2 - \|f(a_i) - f(n_i)\|_2^2 + \alpha)$$

**Triplet-Mining** From the fact that we learn with triplets, some questions arise: How do we find the triplets and how do we feed them to the network?

The process of finding triplets is referred to as triplet-mining (Schroff et al., 2015). We have two different options of mining: online and offline.

With offline mining we search through all our data, find all possible triplets and feed the triplets to the network batch-wise. Note that in order to feed triplets to the network, we actually need three copies of the same network whose outputs we then combine to compute the loss. This is also why the architecture is called a Siamese Network. This strategy ensures that we actually use all triplets for learning. (cf. figure 7)

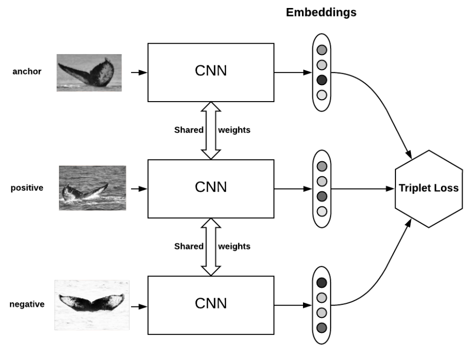


*Figure 7.* Offline-Triplet-Mining (Moindrot, 2018)

However offline mining has major drawbacks. Firstly, precomputing all triplets can become infeasible for larger datasets. Secondly, as more and more triplets satisfy condition 1, their loss becomes zero and thus they have no use for learning our model. In this way, we waste lots of computing power calculating the embeddings of these triplets.

This is where the idea of online mining sets in. Instead of precomputing all triplets, we calculate the embeddings for a minibatch of data, and then find triplets within the minibatch (cf. figure 8). Triplets which have no influence on the loss, can thus be omitted without additional computation. On the other hand, we sacrifice the ability to learn with all possible triplets. This problem can be counteracted by choosing a large batch-size.

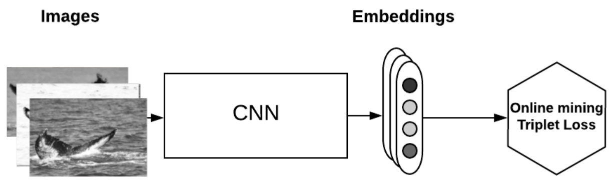


*Figure 8.* Online-Triplet-Mining (Moindrot, 2018)

**Triplet-Selection** We also have different options of further screening our triplets. What naturally comes to mind is to pick triplets that maximally violate the constraint 1. Or more precise, to only pick triplets where

$$\|f(a_i) - f(p_i)\|_2^2 > \|f(a_i) - f(n_i)\|_2^2 \qquad (2)$$

These triplets are called hard triplets. One might argue that hard triplets often contain outliers that are not beneficial for the learning process, however they also reduce bias towards very similar anchor and positive pairs. Schroff et al. propose another policy called semihard triplet selection where only triplets are considered that lie within the margin $\alpha$. The different triplet-selection policies are visualized in figure 9.

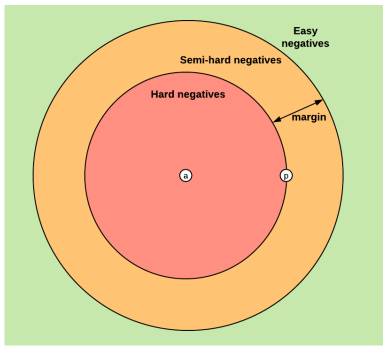


*Figure 9.* Different triplet selection policies (Moindrot, 2018)

#### 5.2.2. Applied strategies

**Top-Model** In our final model, we decided to use online triplet mining because of limited time and computation ressources. Furthermore, we employed the hard triplet selection policy, because we found in practice it leads to a much faster loss convergence than the semihard variant. One reason for that might be the occurence of pseudo-duplicates within the training set. If these very similar pairs of images get selected as anchor and positive often, the learning effect is minimal and it might lead to overfitting. The hard triplet selection avoids this problem. In order to enable proper triplet mining, we chose a large batch-size of 800 examples during training. If the computing resources are available, an even larger batch-size of a few thousand images is advantageous (Schroff et al., 2015).

The CNN-Architecture used for the Top-Model is almost the same as the one employed in the regular Convolutional Network approach. A detailed description can be found in table 2. Note that in the last layer a L2-normalization is applied to the embedding such that $\|f(x)\|_2^2 = 1$. This was also done in Koch et al. and Schroff et al., but no reasons were given for why this is necessary. We suspect this normalization limits the amount of possible solutions in such a way that the number of local minima is reduced. In practice we found that the L2-Normalization leads to a better validation accuracy. Furthermore, we have used the $tanh$ activation function to allow for negative values in the embeddings and therefore encourage a further spread of the embeddings on the hypersphere. Moreover, a large dropout rate of 80% was applied to the additional layer, in order to prevent overfitting. For lack of computational ressources, these parameters were not optimized by a full exhaustive grid-search, but rather carefully selected by hand.

| **layer** | **size-out** | **activation** | **params** |
|---|---|---|---|
| input | 320x205x1 | - | 0 |
| inceptionV3 | 2048x1 | - | 0 |
| dropout | 2048x1 | 80% | 0 |
| dense | 256x1 | tanh | 524,544 |
| l2-normalize | 256x1 | - | 0 |
| total | | | 524,544 |

*Table 2.* CNN-Architecture of Siamese Network (Top-Model)

**Bottom-Model** With the Triplet-Loss, the Top-Model is trained in way, such that euclidean distance between the embeddings directly corresponds to image similarity, or in a perfect world, the similarity between the two flukes depicted in the images.

Predicting the class of a new picture thus becomes a Nearest-Neighbour Problem. We calculate the embeddings for all training data and fit a Nearest-Neighbour classifier. Accordingly, the predictions for a new image can simply be obtained from the nearest neighbours of that image in embedding space.

The approach that was employed in the final solution, is to find the five nearest neighbours with unique classes from the training set and to order them according to their proximity.

We found that more complex variants of KNN like majority vote among the nearest neighbours are of no use for the problem at hand, because in most cases we only have a single example for each whale in the training set.

In order exploit the high rate of ”new_whale” images in the test set, we introduced a distance threshold, that can also be seen as a certainty cutoff. If the distance to the nearest neighbour lies beyond that threshold, the new image will be predicted as ”new_whale” in the first spot.

## 6. Evaluation and Comparison

The two models were compared by uploading a submission file containing the label predictions for the test set to the challenge website. Submissions are evalutated according to the Mean Average Precision @ 5 (MAP@5):

$$MAP@5 = \frac{1}{U}\sum_{u=1}^{U}\sum_{k=1}^{min(n,5)} P(k)$$

where $U$ is the number of images, $P(k)$ is the precision at cutoff $k$, and $n$ is the number predictions per image. Table 3 shows how the different models performed on the test set.

| **Model** | **MAP@5** | |
|---|---|---|
| | Cropping | No cropping |
| Dataset Exploit | 40.8% | 40.8% |
| Siamese Network | **45.2%** | 44.6% |
| CNN | 14.8% | 12.6% |
| CNN + Exploit | **46.5%** | 44.2% |

*Table 3.* Test performance of different models

As described in section 3.1, a score of 40.8% can be reached by exploiting the large rate of ”new_whale” and duplicates in the dataset. Contrary to expectations, the regular CNN architecture slightly outperformed the Siamese Network, however only so when combined with the abovementioned exploit. Nevertheless we see more potential in the Siamese architecture. In the following sections we will discuss the advantages and disadvantages of the two models.

**Siamese Network** In general, Siamese networks have some remarkable advantages. They are able to learn from just a few samples per class (Koch et al., 2015; Schroff et al., 2015). Also the architecture allows for an arbitrary number of classes, because the Top-Model just predicts similiarity instead of class probability. In this specific data set the implicit detection of duplicates by the Siamese network is another convenient advantage. Since duplicate images always have the same embeddings, the euclidean distance to original will be zero and they will thus be predicted correctly.

Despite all these advantages, there are also drawbacks when using the Siamese architecture. Several factors exist, that contribute to higher complexity compared to a regular CNN. First of all, we have two models instead of just one (Top-Model and Bottom-Model). The Triplet-Loss function used in our Top-Model is quite complex itself compared to other loss functions. In addition, it requires triplet mining which again is not a straight forward process and can change the final results significantly when done in different manners. Unwanted biases can easily emerge accidentally, when not considering these details carefully. Therefore it is comparably hard to optimize this model, because many parameters need to be considered and tested.

**CNN** The main advantage of the classical CNN architecture is its simplicity compared to the Siamese architecture. The architecure can be optimized easily by just choosing an adequate pretrained model and varying the number of layers. Also the validation process is much leaner, since we just need to feed the validation data to the network. Contrary, when using the Siamese network, we need to create and store all training embeddings first, in order to compare the validation embeddings against them afterwards. When it comes to the application of classical CNNs in this project, there are major drawbacks we want to point out here. One main issue is the fact, that there is no trivial way to add new classes to the data set. Adding new whales in the future would require to change the layout of the network (since the last dense layer would need additional neurons). Another disadvantage specific to the given challenge is that due to the forgetting nature of CNNs, it is not guaranteed that the network will classify duplicate images correctly. For the same reason the CNN Architecture is not suited for Few-Shot learning tasks.

## 7. Summary / Discussion

In this part, we will discuss ideas to improve this project even further. This section is based on the notebook of the winner of the kaggle challenge (Martin Piotte). The approach described is a custom model, build completely from scratch. In the following, we will summarize the suggested approach. Instead of the Triplet-Loss function, a binary cross-entropy objective is used (Koch et al., 2015). Instead of feeding triplets to the network, pairs of images are evaluated. A pair might contain two images from the same whale, or images of two different whales.
For this setup it might be helpful to use a different distance function for the embeddings than just the euclidean distance.

It might be a good idea to pay more attention to some specific features of the embeddings. High values for example could be more relevant than zero values. Since we do not precisely know how to choose important features we can train a small network which will learn this for us. Therefore a combination of distance measures is introduced. For each feature compute: the sum, the product, the absolute difference and the difference squared $(x+y, xy, \|x-y\|, (xy)^2)$. These four values are then fed into another small neural network, which evaluates how to weight exact matching zero values, and similar (but not equal) features with high values. The biggest potential to improve the model accuracy seems to lie in the construction of the training data. When mining the image pairs (or triplets) to present to the network it might be useful to carefully consider how often one image should be used as a positive sample and how often as a negative sample. If one image is used only once or twice as a positive sample, but many times as a negative sample, the network might just learn to predict a mismatch every time this images is processed, instead of recognizing the whale in the image.

### 7.1. Key Takeaways

One key takeaway from this project is the importance to (manually) examine the data in depth. Doing so allows for detection of parts of the data set which are not useful for the training process. Reasons to exclude samples from the training data might be the occurrence of extreme outliers or corrupted data. Especially when using the hard Triplet-Loss outliers impact the results strongly as pointed out before (cf. section 5.2.2).

Additionally we want to highlight the importance of creating an unbiased and balanced training data flow, even more so when using the Triplet-Loss function. In our case this means to ensure the that samples are used as positive examples and negative samples equally often. More generally speaking this means to avoid any kind of bias in the training set as far as possible.

These two basic steps (selection valid data and creation of a meaningful training set) have a significant impact on the final model performance.

Concludingly, we emphasize that preparing and cleaning the data has been at least equally important as choosing a suitable network architecture and optimizing the parameters.

## References

deepsense.ai. Which whale is it, anyway? face recognition for right whales using deep learning, 2016. URL https://deepsense.ai/deep-learning-right-whale-recognition-kaggle/.

Kniest, E. and Burns, D. Fluke matcher: A computer-aided matching system for humpback whale (megaptera novaeangliae) flukes, 2010. URL http://old.whaleresearch.org/articles/flukeMatcher.pdf.

Koch, G., Zemel, R., and Salakhutdinov, R. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

Li, X., Chen, S., Hu, X., and Yang, J. Understanding the disharmony between dropout and batch normalization by variance shift. *CoRR*, abs/1801.05134, 2018. URL http://arxiv.org/abs/1801.05134.

Moindrot, O. Triplet loss and online triplet mining. https://omoindrot.github.io/triplet-loss, 2018.

Piotte, M. Bounding box model, 2018a. URL http://www.kaggle.com/martinpiotte/bounding-box-model.

Piotte, M. Whale recognition model with score 0.78563, 2018b. URL https://www.kaggle.com/martinpiotte/whale-recognition-model-with-score-0-78563.

Ranguelova, E., Huiskes, M., and Pauwels, E. J. Towards computer-assisted photo-identification of humpback whales. In *Image Processing, 2004. ICIP '04. 2004 International Conference on*, volume 3, pp. 1727–1730 Vol. 3, Oct 2004. doi: 10.1109/ICIP.2004.1421406.

Schroff, F., Kalenichenko, D., and Philbin, J. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Toumbourou, L. Humpback whale id: Data and aug exploration, 2018. URL https://www.kaggle.com/lextoumbourou/humpback-whale-id-data-and-aug-exploration.