UNIVERSITÄT OSNABRÜCK

Happywhale - Automated Marine Animal Recognition

Louis Kapp, Felix Hammer

Abstract
As the urgency of nature conservation rises daily, see it as our responsibility to support this endeavor by means of technology. The Kaggle Happywhale challenge, with the goal of individual whale recognition, provides the perfect opportunity to this using machine learning. In this paper, we investigate the usefulness of combining Softmax classification with a semi hard triplet-loss approach. We also test performance of several Convolutional Neural Network architectures. Our test findings conclude that InceptionV3 is the best performing architecture, although combining Softmax and Triplet loss is not a useful approach (in spite reduced convergence time). It turns out that our ”lazy” approach is not sufficient for obtaining good results. A likely cause is our usage of semi-hard triplets, while the dataset contains many hard triplets.

Keywords
Triplet loss, Nature Conservation, Machine Learning

1. Introduction

As climate change and environmental pollution intensify, nature conservation becomes increasingly important. A part of nature conservation is also the study of animal behavior, migration routes and population density, to better understand the problem and obstacles certain species are facing, so we can tackle and prevent these problems to the best of our abilities. To do this, we must be able to differentiate between individuals of a species. In humans, this is easily done by face or fingerprint recognition. But what about animals?
To date researchers manually differentiate them by the shape and markings on their tails, dorsal fins, heads and other body parts. This is time consuming and difficult work, since it takes the eye of a good researcher and much time to identify, match or tell individual animals apart.
Hence the question arises: When we can use automated identification for humans, is there a similar approach possible for animals?
The recent advances in facial recognition were mostly commercially motivated. Photo identification for animals seems like a less lucrative endeavour which moves it out of the research limelight. However, a technique like this could simplify the analysis of wildlife and therefore nature conservation significantly.
This is why we decided to take on the Kaggle Happywhale challenge (Kaggle, 2022). This goal of this challenge is to train a machine learning model to identify whales and dolphins individuals. The competitions winning model will be used on happywhale.com (n.d., 2022), a research collaboration platform which aims at increasing global understanding and caring for marine animals.
We want to use this challenge to look for relevantly easy approaches that could be implemented by conservation organizations to automatically detect individuals on their data-sets as well.

2. Kaggle Happywhale Challenge

In the following chapter, we are explaining the Kaggle Happywhale challenge. This includes goal-setting and a comprehensive analysis of the dataset.

2.1 Goal

The Happywhale challenge is a research prediction competition, open to everybody with a Kaggle account. It’s goal is to build a machine learning model which can reliably recognize individual whales and dolphins. The model should also be able to classify individuals it has never seen before as ”new”. Such a model would save experts, who - up until now - have to analyse the images manually, a tremendous amount of work.

2.2 Data

The data (Kaggle, 2022) to be used for this challenge is split in to training and testing data. The training data consists of 51 thousand JPEG images of whales and dolphins. The training images are labeled with the according species and a unique individual identification string. The testing data consist of 28 unlabeled images containing some new whales and dolphins which are not present in the training data.
In total there are about 15 thousand different individuals and 30 different species. This data was manually curated by researchers from all over the world.
With a median image shape of approximately (3000,1500), most images have high resolution. This makes unique pat-

Figure 1. Typical images from the data set.

terns and markings of each individual identifiable by a neural network, even though most images only contain small parts of the marine animals such as dorsal fins and parts of the backs.
Some further analysis shows that 59% of all individuals have only one image of themselves. They make out 18% of all images in the data set. 5% of all individuals have at least 10 images of themselves. These are 47% of all images in the data set.
This is a large unbalance in number of images per individual and will make training of the network much harder.

3. Model selection

At first glance, one could think that the problems like the Happywhale challenge are generic multi-class classification task. This would mean that the model should be able to classify an input image of an individual correctly as just this individual. For tasks like this, the default approach is the following:
Feed the input image to some Convolutional Neural Network (CNN) architecture which outputs an embedding of this image. This embedding can then be flattened and fed into a Dense Layer which will output the predicted class. (Schmid, Horn, Lange, Pink, & Kobrock, 2021b)

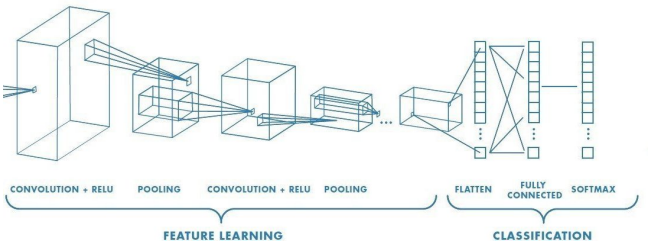

Figure 2. Vanilla image classification (Raghav, 2018).

3.1 CNN architecture

As a first step of model selection we compare popular CNN architectures that we want to test. It is often a good idea to bet on architectures which already solidified their position on the grandstand of the machine learning domain of interest. Therefore we had a look at the most implemented models for image classification tasks.
Looking at this list we decide for a Residual Neural Network (ResNet), a Densely Connected Neural Network (DenseNet), an EfficientNet and an InceptionNet. (Meta Platforms, 2022)
All of these networks are notorious for being capable to classify images with very high accuracy while being somewhat computationally efficient (at least compared to other machine learning approaches).
Additionally, there are countless pre-trained models with easy access points available for these architectures. Especially the models pretrained on ImageNet (Deng et al., 2009), one of the world's largest labeled image databases, are extremely powerful for image classification tasks and therefore of great interest to us.
We decided to try out the following architectures with pre-trained weights from ImageNet: ResNet50, ResNet50V2, DenseNet101, EfficientNet and InceptionNetV3. We also trained ResNet50 ourselves. More about that can be read in Softmax Model.

3.2 Softmax activation

The activation function of the last Dense Layer in this pipeline is usually a Softmax function (Schmid et al., 2021a), which computes a probability distribution σ over the image membership for each class K.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1,\dots,K \quad \text{and} \quad (z_1,...,z_K) \in \mathbb{R}^K$$

K is determined by the constant number of neurons in the final Dense Layer. Unfortunately, we are working with a dynamically changing number of classes - we do not know every individual whale in the ocean (let alone having images of them).
For now, this eliminates the Softmax activation function as a suitable approach to our task.

3.3 Triplet loss

Luckily, the domain of metric learning offers us an alternative approach which is more suitable for our problem: Triplet loss - originally presented in the FaceNet (Schroff, Kalenichenko, & Philbin, 2015) paper.
The goal of triplet loss is to learn an embedding space in which similar sample pairs stay close together and dissimilar ones are far apart. It does this by computing a distance between an anchor image a and a positive sample p, as well as between a and a negative sample n. a and p are different images of the same individual, while n is an image of a different individual.

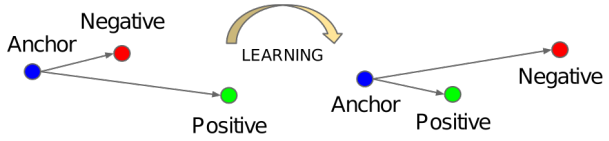

Figure 3. Triplet loss visualized (Schroff et al., 2015).

The learning goal is then to minimize the distance between a and p and maximize the distance between a and n at the same

time (Weng, 2021). Triplet Loss for some triplet (a, p, n) in the embedding space is defined as:

$$\mathscr{L}_{\text{triplet}} = \max(0, d(a,p) - d(a,n) + \varepsilon)$$

where d denotes the L2 norm (calculated from the euclidean distance between the embeddings) and the margin parameter ε denotes the minimum offset between distances of $d(a,n)$ and $d(a,p)$. This prevents the the network of gaming the function and circumventing its actual intention. If there was no ε, the network could just map the complete data set onto the same point in the embedding. This would cause all distances and the loss to be 0. The margin ensures, that in this case, the loss would still be positive. (Kha Vu, 2021)

3.3.1 Triplet Mining

Of course, triplet loss is far from being a perfect method of choice as well. One of it's main limitations is that during one comparison only one negative example n is compared with the anchor a. The disregard of all other n could lead to an embedding space where some dissimilar pairs are still in close proximity to each other - just because they were not compared against each other. (Sohn, 2016)
This is why, in order to train a triplet loss model successfully, one has to pay close attention to the composition of each triplet. If we want the model to learn something we should pick an n that is not obviously a different individual than a.

To do this, we could choose a challenging n that is closer to the a than the positive sample p.

$$d(a,n) < d(a,p)$$

This is called *hard triplet mining*. In theory, it should guarantee optimal learning success. However, it is prone to get stuck in local minima and the samples are rather hard to select from our data set.

An approach which tries to solve this is *semi-hard triplet mining*. Here the triplets are chosen in a way, that n is not closer to a than p, but the function still has a positive loss.

$$d(a,p) < d(a,n) < d(a,p) + \varepsilon$$

The distance between a and n is still in the range of margin ε. This way, the model can still learn but is more unlikely to end up in local minima.

We need to specify the triplet sampling process even further by choosing between *offline* and *online* triplet mining.
With *offline mining*, the triplets are generated at the beginning of each epoch. The embeddings are computed on the training set and then only (semi-) hard triplets are selected. This technique is rather inefficient because we need to do a full pass on the training set for each epoch and update the triplets.
Doing *online mining*, samples are produced for each batch of inputs. Within a batch of B examples we can find a maximum of B^3 triplets. Many of these triplets are not of shape (a, p, n) and are therefore not valid. Still, it does not require updating all samples offline and more samples are created for a single batch. This makes the approach way more efficient (Moindrot, 2018).

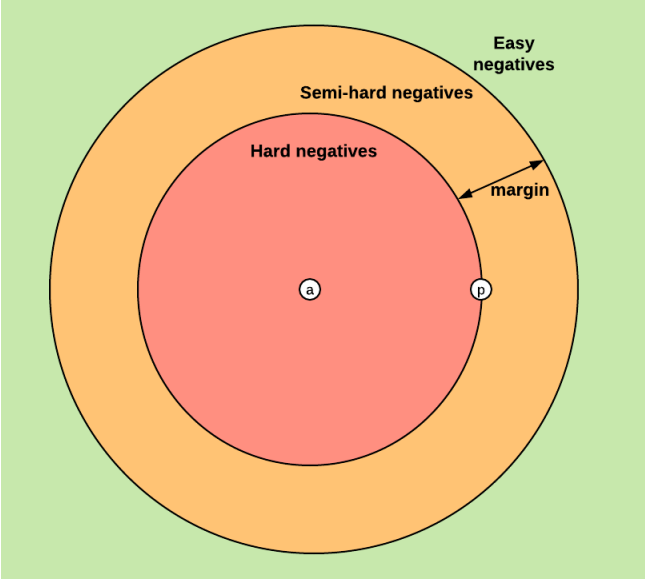

Figure 4. Different kind of Negatives visualised (Moindrot, 2018).

3.3.2 Shortcomings

There still remain problems with the triplet loss function and contrastive learning approaches in general.
It is hard to guarantee that embeddings of the same individual will be pulled together in euclidean space (expansion problem). In addition, there is the so called sampling issue (Kha Vu, 2021): Online triplet mining is hard to implement for very unbalanced data sets like our Happywhale data set.
There are modern alternatives to triplet loss such as Cos- and ArcFace, which are based on angular distance and margin. They (partially) solve the above problems but we nevertheless wanted to try out vanilla triplet loss and see how far we will come with this approach. Since our paper should also be of some educational value and explain the concepts well to other students, triplet loss is also more suitable because it is less advanced and needs fewer prerequisites to understand.
Because of the mentioned advantages in Triplet mining, we also decided to implement online semi-hard triplet mining.
There is still one disadvantage of triplet loss which we did not talk about up until now: Evaluation of training progress is computationally expensive. As we want to evaluate the usefulness of the different CNN architectures, we need to compute an evaluation metric (i.e. accuracy) on the validation set after each training epoch. To do this, the embedding of every image from the validation set needs to be compared to all embeddings of the training set. This means there have to be many point-wise distance calculations in a high dimensional

euclidean space calculated. To be specific, we need to do $37598 * 4177$ computations in a 256-dimensional embedding space for each epoch. Furthermore, the convergence time for triplet loss models is said to be rather high.
Doing this for more multiple hyperparameter settings for each architecture is not feasible with our computational resources. Is there a cheaper workaround?

3.4 Combining Softmax and Triplet Loss

We already found out that the Softmax activation function is not useful for our final model but maybe we could still utilize it for some architecture comparison. Computing accuracy of a Softmax based model is a breeze compared to models based on contrastive loss.
We can imagine that an architecture that performs well over a finite-class classification task using Softmax is also likely to perform well for a classification problem with a dynamical number of classes like Happywhale. So why not try this out? Our approach will now look like the this:
We will train the said models with a fixed size Softmax Dense layer at the end on the different (fixed) whale species. During this training process, we evaluate their performance on the validation set, taking accuracy and convergence speed into account. After training is finished, we compare their performance on the test set and select the most robust and promising architecture for our final triplet loss model.

The final model will essentially be a Siamese Network which is composed of 2 identical CNN architectures that share their weights. The according inputs are the sample pairs (a, p) and (a, n) which are then projected into the embedding space. As explained earlier, the triplet loss then should ensure reasonable euclidean distances between the embedding pairs.

4. Hypotheses

Knowing all this, we can motivate the following hypotheses that we want to investigate further:

1. Is a ”lazy” approach of just using prebuild & pretrained architectures and preconfigured loss functions sufficient enough to produce significant results in individual classification?

2. Is Softmax classification training success a useful indicator for triplet loss training success? To be specific: Does the best performing CNN architecture under Softmax also show best performance of under triplet loss?

3. Can you reduce triplet loss training time with network weights pretrained on Softmax classification?

5. Data Pipeline

5.1 Preprocessing

We apply several preprocessing steps to get the data into an optimal shape for our model. We also create One-hot labels.

5.1.1 Normalization

Normalization ensures that all images have same format.
One of the first steps is downsizing the images. Although our mostly high resolution pictures allow to see patterns and unique marks of the whales in great detail, we have to downsize the images enough that our memory can handle it. It is also useful to have every image in the same quadratic shape. We chose the shape (224, 224) since it is little enough to not run into memory constraints but still large enough for the model to recognize some details. The pretrained ImageNet-models also use this input size. The automatic interpolation of Tensorflow ensures that the images still look somewhat natural and not too distorted.
We also converted the pixel values from integers to floats to make sure that Tensorflow can work with them. The float values were fixed between -1 and 1.

5.1.2 Foreground Extraction

Since the images often contain much ocean in the background and the model should definitely not take this as feature, we thought it would be a good idea to crop the background such that only the whale on white background is in the image.
We mainly used preexisting algorithms from the Kaggle Humpback Whale challenge (Kapse, 2020) for this. Unfortunately, large parts of whales were cropped in many images.

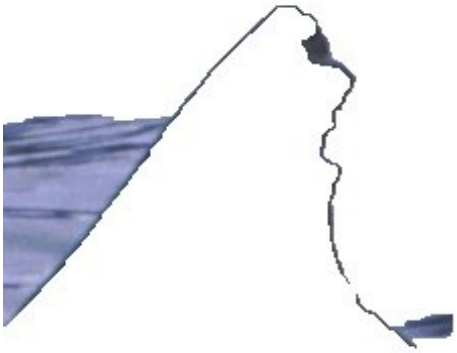

Figure 5. Failed foreground extraction.

This is why we decided not to include this preprocessing step in our pipeline.

5.1.3 Data Augmentation

Data Augmentation has the goal of increasing the size of the data set.
This is especially useful for obtaining multiple images of whale individuals, for which only one or two images exist. However, we decided against data augmentation because it significantly impacted training speed - even tough we might have obtained better results with it.
We still decided to incorporate classic augmentation steps, such as random image flipping, contrast and brightness, to

introduce more randomness to our data. This might lead to a more regularized model performance. Being nit-picky, these steps also count under "normalization", but we wanted to lay down our reasons for not doing data augmentation in a separate paragraph.

5.2 Dataset creation

Since online mining samples the triplets within one batch, we have to ensure on batch creation that there are sufficiently many different pairs of anchors and positives per individual within a batch. As analysed earlier, there are many individuals with only one or two images in the Happywhale data set.
Suddenly, this turned out to be a quite interesting constraint satisfaction problem (CSP).
First we had to consider what to do with the individuals with only one image, which make out 59% of all individuals and 18% of the dataset. One approach could be to use image augmentation and then pair the individuals with a altered version of their images. But since our model needed way to much time already, we decided to use these individuals not for training, but for another process later.

5.2.1 Smart Batches Algorithm

To solve this CSP, we came up with an algorithm which can easily be generalized for other triplet loss applications.
First and foremost, we decided to only accept an even batch size, which relaxes the CSP quite a lot.
We choose 64 as a batch size, simply because it yielded the best performance - in terms of training time - on our setup (Nvdia GTX 1080).
Let's say we have N images of different animals, with at least two images per animal. We also have an even batch size b. We need to divide by the number of batches, which is $\lceil N/b \rceil = M$. The last batch will not have size b but rather $M\%b = L$. Now we have to distribute our N images over these batches in such a way, that there is never a single image of an individual in a batch and that one batch never contains images of only one individual.

The way we solve this problem is by creating *two separate pools*. In the one pool we put in all the images of animals with an *even* number of images and in the other all the animals with an *uneven* amount of images. Then we iterate through every animal in the *uneven* pool.
We will take the first *three* images of every animal and keep them in the *uneven* pool. The *even* amount of rest can be thrown into the *even* pool.
For example: If a whale has 17 images, we will keep the first 3 pictures in the even pool and put the next $17-3=14$ pictures into the even pool.

Now we have only triplets left in the *uneven* pool.
For a healthy amount of randomness, we will shuffle them around. For reproducibility, we set a seed beforehand.

There is a small special case when N - and subsequently L - is *uneven*. We know that the *even* pool still contains an *even* amount of images. So the only source for the *unevenness* of N can be in the *uneven* pool. This would mean that we have an *uneven* amount of triplets. In this case we will just take the first triplet and put it into the last batch.

We have enforced that:

1. Every batch has an *even* amount of space left.
2. There is an *even* number of images in the *even* pool
3. There is an *even* number of triplet-pairs in the (initially) *uneven* pool

We have solved the CSP with this algorithm:

1. Combine the triplet-pairs to pairs of 6 each.
2. Distribute them over the batches.
3. Form positive triplet-pairs of 2 in the even pool and shuffle them (with seed).
4. We fill up the batches with the triplet-pairs of 2.

Because we have at least two or three images of every animal in a batch, we know that that there is never a single image of an individual in a batch. Because we shuffled the first pairs of two, one batch most likely never contains images of only one individual. If this is not the case, we can simply choose another seed. To now keep this order, we will make sure to not shuffle not to shuffle at a later step in our pipeline.

6. Our Softmax Model

For training the Softmax model on species classification, we decided to use all available training images. For Softmax standards, 51000 images are not that many. This is why we decided for train/validation/test split of 0.8/0.1/0.1.
Since we wanted to train, validate and evaluate or model on predicting all 30 species, we wanted to have at least one image of every species present in every dataset. To achieve this, we used the Smart Batches Algorithm, simply because it likely to create such a split.
We called all functions with different seeds and quickly found the first valid candidate "5", which we then used for all our Softmax models.

6.1 Configurations and Hyperparameters

We loaded all architectures with the *include_top* setting as *false* and set *pooling* to *max*. We then simply put a dense layer with 30 neurons for the species and a Softmax activation function on top. We choose *Adam* as an optimizer with a *learning rate* of 0.001 and set *Categorical Cross Entropy* loss. Nothing special - pretty standard image classification settings.

6.2 Evaluation

6.2.1 ImageNet vs Random Weights

We first wanted to test whether loading our models with weights, which where pretrained on the ImageNet dataset, would lead to a faster convergence. To do this, we trained a ResNet50 for 10 epochs each. Once with the pretrained ImageNet weights and once with randomly initialized weights. The ImageNet weights beat the other in every metric.

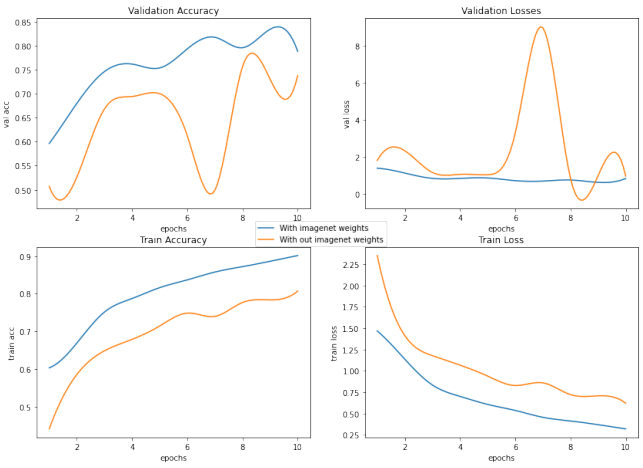

Figure 6. ImageNet vs random performance.

Because of this, we then decided stick to models with ImageNet weights for our further analysis.

6.2.2 Comparing CNN architectures

Further, we test the performance of several CNNs. As mentioned in CNN architecture, we do not want too many parameters so we could still train them locally. We tested all architectures for 10-15 epochs.

These were the results:

Architectures	ACC_{10}	$s/epoch$	ACC_{10}/time
ResNet50	81,9	532	0.15
ResNet50V2	85,4%	411	0.21
DenseNet100	89,7%	506	0.18
InceptionNetV3	90%	271	0.33

Table 1. ACC_{10} denotes the validation accuracy after 10 training epochs. s =seconds.

We also tried to use several versions of EfficientNet, but sadly our kernel always died pretty quickly.

As you can see, InceptionNetV3 had - compared to the others - an outstanding performance considering training time. This is why we selected it to be the model, which we will train further and of which we will recycle the weights later for the Siamese Model Training.

We then decided to train for 70 epochs in total and evaluate the best model checkpoint with the best validation accuracy on our test dataset. We achieved an test accuracy of 94.45% and a k3-accuracy of 98.79% on our test data, which was quite impressive.
We then analyzed the accuracy for every species and found that as one might expect, the model was really good at recognizing species with many images, while struggling with the uncommon ones.
For example, it completely failed to detect the least common species (”Frasier Dolphins” with only 14 images). Strangely, the best performing species were the ”Commersons Dolphins”, with only 90 images in the database. We assume this has to do with the unique, black and white look of their dorsal fins and the very high image quality of the images, which you can see here.

7. Our Triplet Loss Model

7.1 Specific Dataset

Since the Kaggle Challenge already provides a testing mechanism, we only had to choose a validation dataset. We already disregarded more then half of individuals by now so we do not want to throw away more data. Hence, we decided to only consider individuals with > 3 images. We then used a method very similar to the smart Smart Batches Algorithm:

1. Iterate over all the individuals with > 3 images.
2. Take the first two images and put them to the side to make sure we don’t disregard individuals.
3. Shuffle the remaining images (with a seed).
4. According to the split ratio S (we choose 0.1), take the first $\lceil S * len(remaining\ images) \rceil$ individuals as your validation dataset.
5. Use all the other images as the train dataset.

Then we called the Smart Batches Algorithm and do the remaining standard preprocessing steps. We used the seed 0.

7.2 Configurations and Hyperparameters

To test our hypotheses we will train these models:

1. InceptionV3 Model with the weights pretrained on the Softmax species.
2. InceptionV3 with imagenet weights.
3. Resnet50V2 with imagenet weights.

For simplicity, we will refer to the models from now on as ”Species-Weights”, ”ImageNet-Weights” and ”Control-Model”.
As for the embedding block, which we put on top of the CNN, we choose this architecture: 3 Dense-Layers with 512, 256 and 256 output neurons respectively and *ReLu* - activation

functions. We then choose to use a l2-normalizing layer as our final output layer to enforce more evenly distributed distances to then later be able to better identify new individuals. We do automated online semi hard triplet mining with a default margin of 1.
Since the the Kaggle Challenge offers a nice automatic way of testing models, we had to only come up with a validation procedure for our inter-model comparison. We decided for this 2 stage validation procedure:

Since our model should learn to 1. recognize and 2. also recognize when it has not seen an individual before we come up with this 2 stage validation procedure:

Firstly since our model should learn to recognize individuals it has seen before, we decided to measure the accuracy of recognizing the a known individual, the k5-individual accuracy and the accuracy of classifying the right species. To achieve this we firstly we calculated the embeddings of our train and validation datasets. Then we computed the pairwise distance matrix with an euclidean metric of the two embeddings and sort out the labels with the 5 closest values. With these labels we now could calculate the beforehand mentioned accuracies. In addition we used UMAP to visualize a dimensional representation of our embeddings. Secondly our model should also recognize when it has not seen an individual before. For that we used the individuals with only one picture and calculated there embeddings. We then calculate the pairwise distance matrix of those embeddings with the train embeddings and sorted out the distance to the nearest neighbor. We then compared this distance to with the average distance of individuals which our model is already familiar with. For that we created density plots of the distributions.

7.3 Evaluation

7.3.1 General Training Behaviour

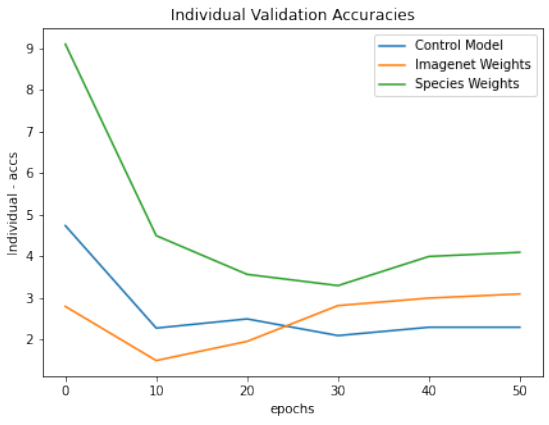

Figure 7. Individual Validation Accuracies

When looking at these results in Figure 7 we were really confused at first. Why does every Model has it's best performance before training? After doing some did some data-analysis we had this idea: Before training all the weights of our embedding block are randomly initiated. Thus a high Validation Acuraccies arises probably from the very uneven distribution of individual counts. First we have to look at the dataset which we use for our siamese model training and validation, the dataset containing all the individuals with at least 2 images. The top 300 individual of this data, make out 5% of all individuals and 40% of all images. So if we know choose to random images from the data the chances of getting the same individual are quite high. Interesting is now the composition to which species these individuals belong:

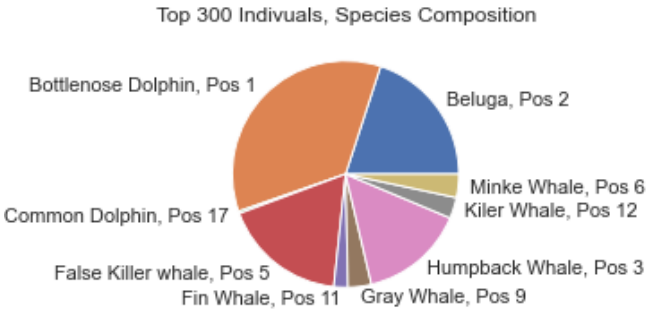

Figure 8. Species Composition of top 300 Indivuals. *Pos* refers to the degree of frequency

In addition what is very interesting is that the species accuracy, the likehood of the next neighbor in the embedding space, is for all models around around 80% after epoch 30. If we combine this with the fact that the top 300 individuals belong mainly the most frequent species, we can refer that the most frequent classes are quite densely packed together in the embedding space. This would mean that most images are in densely packed spheres. We assume that therefore we have a lot "Hard-Triplet-Pairs" in within these spheres of the most frequent species. In addition the 2d UMAP-plots of our embeddings support these hypothesis.

7.3.2 Species Weights vs Imagenet Weights

All the 3 validation accuracies (Individual, k5-Individual, Species) and also the 2d-UMAP-plots before and while training show that the Species Weights Model performs significantly compared to the Imagenet-Model.

7.3.3 Species Weights vs Imagenet Weights

All the 3 validation accuracies (Individual, k5-Individual, Species) and also the 2d-UMAP-plots before and while train-

ing show that the Species Weights Model performs significantly compared to the Imagenet-Model.

7.3.4 Imagenet Weights vs Control Model

Although the InceptionV3 Model outperforms the Resnet50V2 model in terms Individual and k5 accuracy the effect is by far not as strong in the Softmax case. Especially if we into account that it took the InceptionV3 model almost double as long as the Control Model to have a convergent training loss.

7.4 Kaggle Test Dataset Evaluation

Since our best performing model is quite ironically the Species-Model before training even one epoch with the triplet loss, we choose it for the test evaluation. To now test our model we had to create a submission.csv according to this guidelines. Most importantly is that we not only have to identify individuals which we have seen before, but also new individuals. For that we used the density plot of the average distance of new individuals to the their next neighbors compared to known individuals. Looking at the distributions we decided to expect there to be a new individual at a distance of 0.35. To generate the submission.csv we now first had to compute the embeddings of all our data (the Kaggle training data) and the embeddings of the Kaggle test data. Then we computed the pairwise distance matrix and sorted out the 5 closest neighbors and their corresponding labels. For every test image we know iterated through the list of the 5 closest neighbors. If the distance to the neighbor was bigger then 0.35 we inserted "new_individual" into the list at that position. Like this we achieved a score of 0.077.

8. Conclusion

8.1 Hypothesis 1

Is a "lazy" approach of just using prebuild & pretrained architectures and preconfigured loss functions sufficient enough to produce significant results in individual classification?
With a final Kaggle Evaluation Score of 0.077, our model did really not achieve sufficient result to be use full in anyway in marine nature conservation. Therefore we assume that a similar approach is most likely also nut sufficient enough for other domains of nature conservation.

8.2 Hypothesis 2

Is Softmax classification training success a useful indicator for triplet loss training success? To be specific: Does the best performing CNN architecture under Softmax also show best performance of under triplet loss?
With InceptionV3 model with imagenet weights just barely outperforming the Resnet50V2 model with imagenet weights in regard of validation accuracy, we do not have enough data to either support or deny this hypothesis.

8.3 Hypothesis 3

Can you reduce triplet loss training time with network weights pretrained on Softmax classification?
Because of the strong performance of our InceptionV3 Model pretrained on classifying the whale and dolphins species compared to the other models, we have strong reason to believe this hypothesis.

References

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (pp. 248–255).

Kaggle. (2022). *Happywhale - whale and dolphin identification.* Retrieved from https://www.kaggle.com/c/happy-whale-and-dolphin (visited on 2022-04-01)

Kapse, A. D. (2020). *Foreground extraction-opencv.* Retrieved from https://www.kaggle.com/code/akhileshdkapse/foreground-extraction-opencv/notebook (visited on 2022-04-01)

Kha Vu, C. (2021). *Deep metric learning: A (long) survey.* Retrieved from https://hav4ik.github.io/articles/deep-metric-learning-survey (visited on 2022-04-01)

Meta Platforms. (2022). *Image classification.* Retrieved from https://paperswithcode.com/task/image-classification#papers-list (visited on 2022-04-01)

Moindrot, O. (2018). *Triplet loss and online triplet mining in tensorflow.* Retrieved from https://omoindrot.github.io/triplet-loss#triplet-mining (visited on 2022-04-01)

n.d. (2022). Retrieved from https://happywhale.com (visited on 2022-04-01)

Raghav, P. (2018). *Neural network with many convolutional layers.* Retrieved from https://miro.medium.com/max/1200/1*XbuW8WuRrAY5pC4t-9DZAQ.jpeg (visited on 2022-04-01)

Schmid, L., Horn, R., Lange, C., Pink, M., & Kobrock, K. (2021a). *02 training nns.* http://www.studip.uni-osnabrueck.de. University of Osnabrueck. (visited on 2022-04-01)

Schmid, L., Horn, R., Lange, C., Pink, M., & Kobrock, K. (2021b). *05 introduction to cnns.* http://www.studip.uni-osnabrueck.de. University of Osnabrueck. (visited on 2022-04-01)

Schroff, F., Kalenichenko, D., & Philbin, J. (2015, jun). FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE conference on computer vision and pattern recognition (CVPR).* IEEE. Retrieved from https://doi.org/10.1109%2Fcvpr.2015.7298682 (visited on 2022-04-01)

Sohn, K. (2016). Improved deep metric learning with multi-class n-pair loss objective. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 29). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf (visited on 2022-04-01)

Weng, L. (2021). Contrastive representation learning. *lilianweng.github.io*. Retrieved from https://lilianweng.github.io/posts/2021-05-31-contrastive#triplet-loss (visited on 2022-04-01)