Deep Learning Project - INF473V

HappyWhale - Whale and Dolphin Identification

Hugo BOUIGEON, Thomas LI

Table of contents

Introduction

Identifying humans has always been involved in different tasks whether it was for security concerns or for convenience such as organizing photos. To do so, we use biometric identifiers which are the distinctive, measurable characteristics used to label and describe individuals. Different identifiers are used today such as the fingerprints, palm prints, iris recognition, DNA but the most commonly used is face recognition because it doesn't involve any contact and is hence very practical.

However, when it comes to animals, nothing close has been achieved so far. Yet we need it. In marine mammal science for example, it is important to track individual animals over time in order to assess the population status and trends of different species. This is where our project comes in handy. While humans can be identified through fingerprints or facial features, marine animals can be identified through unique markings on their dorsals, fins and tails.

Currently, researchers identify individuals by manual matching, which is a quite tedious and time consuming process. Therefore, the Happywhale Kaggle competition **[1]** provided a dataset of over 50,000 labeled images from 30 different marine mammal species and over 15,000 individuals, which can be used to train deep neural-network models to identify these marine animals. We used this dataset to train and finetune different deep learning models based on already existing architectures such as Alexnet, ResNet18, Resnet50 and EffNets which showed good results with other competitors.

First, due to the complexity of the task, we decided to try to train the models only on the species data. After obtaining satisfying results we moved to training on individuals. We also attempted pretraining our networks and then continued training them on individuals, which showed improved results. The idea behind that was that humans also first learn to classify objects into broader categories, before learning to differentiate more precisely. Lastly we analyzed our results obtained with the different models in order to finetune them even better.

I - Implementation

1) Preprocessing and creating the Dataset

The kaggle competition provided a dataset with over 50,000 jpeg images of different shapes and sizes. In order to use them, we downloaded them all into a folder on the virtual machine, and created a pytorch dataset class to retrieve and transform the images into tensors of size 256 x 256.

We also normalized them to feed them to pretrained models trained on normalized images. This dataset class allowed us to easily create data loaders from which we could get batches of size 16. These batches would then give us the tensor image, the species - labels and the individual_id-labels which were both encoded with unique integers using a python dictionary.

Later, we created separate test dataset and validation dataset classes in order to be able to do fair data augmentation. This separation was stratified on the species by using the train_test_split function from sklearn : we did not want any underrepresented species to be only in the training set or the test set (the least represented species has only 14 pictures).

In order to get faster and better training performances we tried to implement YoLov5 to crop the images at the right spot and resize them directly into a folder, however as YoLov5 is not trained to recognize marine animals, so we would have had to find an external dataset to train it on whales and dolphins. Pressured by time, we decided to download a precropped dataset instead **[3]**.

2) Augmentation

Data augmentation seemed to be necessary since we only had an average of 4 pictures per individual with several individuals with only 1 picture. When we augmented the data, we would augment the training dataset's size by 10. We couldn't do more as the training time would explode. We used the Albumentations library to distort the colors (hue, brightness, saturation) and to apply some shifts, scaling and rotation distortions.

We chose to use small rotation values, because all images were by default taken such that the fin was facing upwards. We chose to not use random-cropping for the data augmentation because the initial dataset contains images of whales and dolphins taken from far away, where the animal is rather small on the image, as well as images where they take up the entire space of the picture. Random-cropping bears the risk of creating input data with no animal in the picture, or only a small part of it.

We also chose not to use horizontal flips. The reason for that is, that individuals are differentiated by small markings on their dorsal fins, which are not symmetrical by nature.

3) Training and Accuracy functions

We implemented a classic training function inspired from the TD6, which we modified so that it could return loss and accuracy graphs of trained models on both the training-set and the test-set. It would also automatically save the model at the end of each epoch to prevent any accidents and to be able to train over them again if necessary.

We also implemented a top-1 and a top-5 accuracy function. We used top-1 accuracy to evaluate our models on species classification and top-5 accuracy to evaluate our models on the classification of individuals, as this was a much harder task and also because this is the Kaggle evaluation metric. Finally, we used Cross-Entropy-Loss as our default loss function.

4) Models

We first implemented a very basic CNN model for species recognition with 2 convolution layers and 2 fully connected layers just to check if everything was working properly. Then, we trained several different already existing models to use transfer learning to speed up the process as the training was very time consuming **[2]**. In order to do so, we just replaced the final classifier with 1 or 2 fully connected layers to give 30 outputs for species and 15587 outputs for individuals. We added a dropout layer as well and initialized the weights with Xavier normal. The different models we tried were : Alexnet, VGG, ResNet18, ResNet50, EffNet_b4 and EffNet_b7 which were all publicly available on the Pytorch library.

EfficientNet (EffNet) has a classic convolutional neural network. But unlike conventional practice that arbitrarily scales these factors, the EfficientNet scaling method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients **[5]**. For example, if we want to use 2^N times more computational resources, then we can simply increase the networks depth by α^N, width by β^N, and image size by γ^N , where α, β, γ are constant coefficients determined by a small grid search on the original small model EffNet_b0.

After some research, we tried to implement a paper on ArcFace **[4]**, which uses an Additive Angular Margin Loss function with a special kind of embedding, but our results weren't satisfying.

5) Accelerating training time

Even though we implemented cuda, training time was pretty slow at first, so we checked what was slowing down the process. To our surprise, it was neither transforming the images nor passing the tensors through the network and applying backpropagation that were slowing down training. The actual step slowing down training was accessing the high resolution image data through the data loader.

So, in order to speed training up, we made the data loader use 6 cores of the virtual machine instead of one, which sextupelt the training speed. In addition, using the already cropped dataset whom's images are 256x256 by default, sped up the data loader by another factor of 10.

6) Analysis tools

We did some occlusion experiments to analyze which part of the images our models actually take into account to classify input images. In order to do so we swept a gray square over the image and marked down a score given to the occluded input on a matrix. The score corresponds to the normalized value of the output for the correct label:

$$score = \frac{output[label] - E(output)}{\sqrt{Var(output)}}$$

Lastly we applied some convolutional gaussian blur on the matrix we obtained this way.

II - Results

1 - Species

Model	TOP 1 Accuracy
Network1	78,8%
AlexNet	18.9%
ResNet18 + 1 FC	92.6%
ResNet18 + 2 FC	91,8%
ResNet50 + 1 FC	94,7%
ResNet50 freeze + 1 FC	84%
ResNet50 freeze + 2 FC	85,4%
EffNet B7	93,2%
EffNet B4	99,2%

These models were all already pretrained **[2]** *and trained with 5 epochs (enough to converge). At each fully-connected layer (FC), there was a dropout layer with p = 0.5 by default except for EffNet models with p = 0.4*

With our first very simple model (Network1) which had 2 convolution layers and 2 fully connected layers, we obtained a 64% accuracy rate after 1 epoch. However training-time for that was 2 hours. Hence, after analyzing the runtime, we decided to work on the cropped YoLov5 dataset. After testing different image sizes as well, we finally found out that image sizes between 64 and 256 did not affect the performance much. However going below would make performance decline and it didn't make sense to go over since the newly downloaded images were of size 256x256. Hence, we decided for the rest of the project to fix the image size to 256. We then managed to obtain a 78,8% success rate in 5 epochs with this first very simple model.

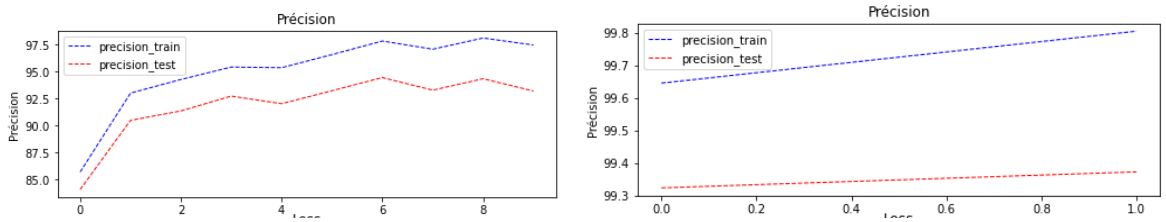
After that, we tried other models using transfer learning but when we used Alexnet we only achieved 18.9%, and its accuracy rate didn't seem to change much between the different epochs. This suggests that this model isn't adapted for our usage here.

With ResNet18 however, we get some satisfying results : 92.6%. We also saw that the number of FC layers at the end doesn't affect the performance by much, maybe because ResNet18 is already deep enough and doesn't need more non-linearity for the classification process and the embedding layers will learn to adapt themselves. Therefore, for the rest of the deeper models, we usually only use one FC for classification.

With ResNet50 we were afraid that because of the depth of the CNN, our model might not converge fast enough so we froze all the layers before the classification. With 1FC we got 84% and with 2FC we got 85,4%. But after unfreezing the layers, we got 94,7% accuracy.

This probably worked because the model was already pretrained and so the weights still weren't too far from the optimal solution.

With EffNet_b7 we got 93,2% after about 7 epochs and with EffNet_b4 we got 99.2% after only 1 epoch ! Therefore, we can conclude that EffNet_b7's size (66M vs 19M parameters) is maybe not so helpful with such an "easy" task.



EffNet_b7 on the left over 10 epochs and EffNet_b4 on the right after 2 epochs

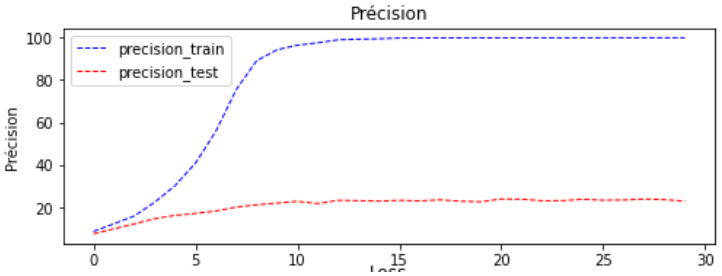
Thus, the main models we will consider for individual recognition are ResNet50 and EffNet_b4 which really seem to stand out for Dolphin/Whale recognition.

2 - Individuals

Models	**TOP 5 Accuracy**
ResNet50 + 1 FC	22,9%
ResNet50 + 1 FC + Data Augmentation x3	24,4%
ResNet50 + ArcFace	1.9%
EffNet B7 + 1FC	28,0%
EffNet B4 + 1 FC (dropout = 0.4)	40,6%
EffNet B4 + 1 FC (dropout = 0.6)	40,3%
EffNet B4 + 1 FC (dropout = 0.8)	11,2%
EffNet B4 + 1 FC + Data Augmentation x10	41,0%
EffNet B4 + 1 FC + Pretrained on species	42,0%
EffNet B4 freeze + 2 FC without Xavier initialization	6,8% after 3 epochs starting from 5,8%
EffNet B4 freeze + 2 FC with Xavier initialization	8,11% after 3 epochs starting from 6,3%

These models were all already pretrained **[2]** *on ImageNet and trained with 10 epochs (enough to converge). At each fully-connected layer (FC), there was a dropout layer with p = 0.5 by default except for EffNet models with p = 0.4.*

ResNet50



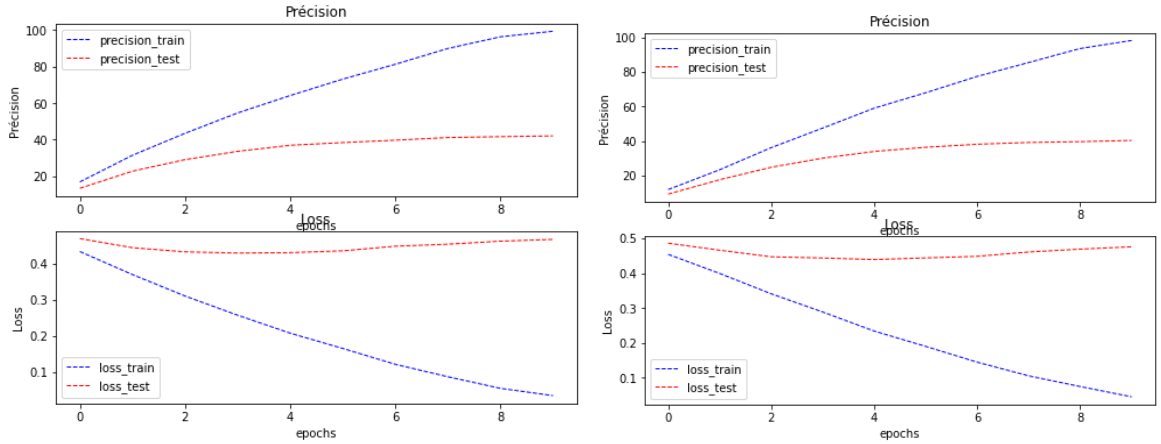
ResNet50 overfitting

With ResNet50 and 1FC layer and 30 epochs, we obtained 22.9% accuracy. We can see on the graph here that overfitting is clearly an issue. At first, we thought that the model only managed to recognize the most represented individuals and memorized by heart all the rest so we tried to do some data augmentation. We obtained 56% accuracy ! However the data augmentation was badly implemented. We augmented the data then splitted the data into the training set and the test set meaning that very similar images were in both datasets. When doing it properly, we achieved 24,4% by augmenting the dataset's size by a factor 3.

EffNetB4

We obtained 40,6% with the default model and p=0.4 for the dropout. We can see here as well that the model overfitted the training set. Hence we augmented the dropout probability to 0.6 and 0.8. However when p=0.8, it was too hard for the model to train going from 6% to 10% in 5 epochs. With p=0.6 we had hope that it would do better since the gap between the training accuracy and test accuracy was smaller at the beginning of the training but it eventually gave a similar result. We also tried to freeze some layers in hope of getting better results but it didn't work out either. Finally, we compared the model with the pretrained model on the species : the accuracy at the end of the first epoch was better (13,4% vs 9,3%) but we also got 42,0% at the end. It is not safe to say that this pretrained model performs better, but it does initialize the weights better and lowers the gap between the test accuracy and the train accuracy at the beginning.

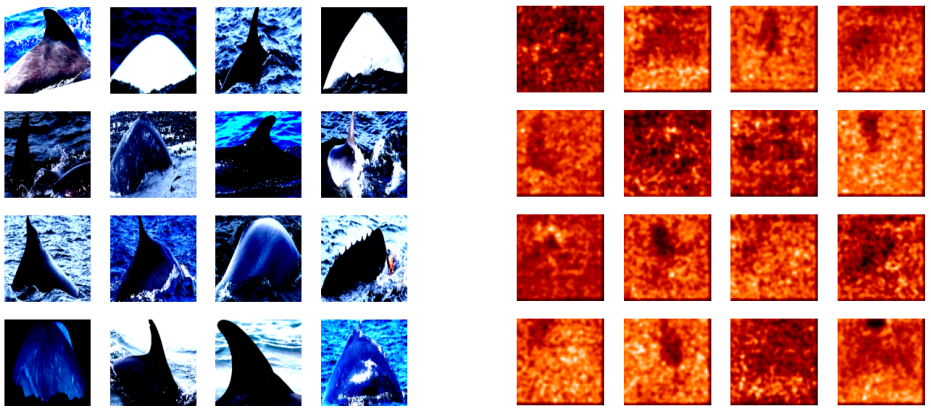
We also tried to augment significantly the data (by a factor 10) but it didn't have a significant impact. This is probably because there are a lot of photos taken in burst mode making our data augmentation less useful since it has already been done in a way.



On the left EffNet_b4 not pretrained on species and on the right EffNet_b4 pretrained on species

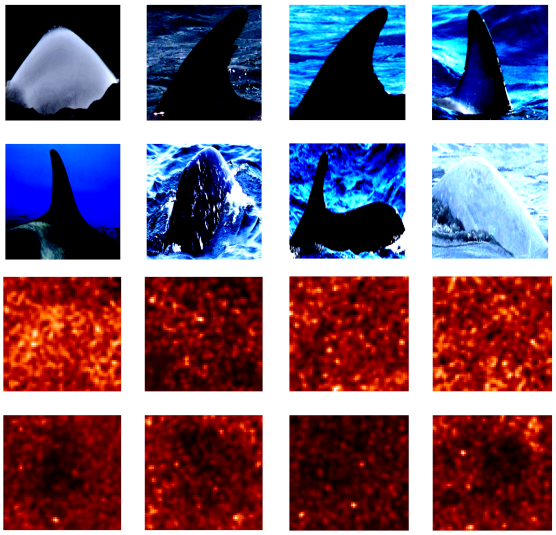
3 - Occlusion maps

To visualize and understand how our models were working, we did some occlusion experiments. Dark regions show relevant information to the models on the pictures.



Occlusion experiments with ResNet50 on species classification

Occlusion experiments show that the network learned in most cases to recognise the whales/dolphins by looking at their fins. In some cases we see that it concentrates on wrong areas though (for example in the last row, third image where it seems to be looking at the surrounding water). We believe that this happens for underrepresented species. Since they don't appear often, the network probably has a hard time learning where to look to find them. Another explanation may be that some photos are taken in burst mode, which leads to the background becoming relevant information to the network for identifying some individuals and by extension species.



Occlusion experiments with EffNet_b4 on individual classification

Occlusion experiments seem to show that the model does seem to look at particular locations, but the formed heatmaps are less precise than with networks trained on species. This is not too surprising as these models have worse accuracy than those trained on species. But it does show a tendency to overfit much more during training, and to learn the photographs by heart.

Conclusion

Our best performing model was the EffNet_b4 model which was able to classify species with a 99.2% certainty and individuals with a 42,0% top 5 accuracy score. Pre-training the network on the species before training it on the individuals, as well as adding a dropout layer has greatly improved the performance of the network and reduced its tendency to overfit the data. However, our occlusion experiments show that our best model still has ways to go to properly recognize individuals, and tends to use the background to classify them, which is a sign of some remaining overfitting issues.

Indeed, our results are still far from competing with some top kaggle competitors. Some implementation ideas which we could have used to our models do come to mind.

For one, we regret not having had the time to properly implement a yolo network ourselves. Furthermore we could have used Cross Validation during training time. Our ArcFace implementation was also lacking and probably one of the defining factors to get better results. We also thought of making some kind of adaptive augmentation which would have augmented underrepresented individuals more than others.

Acknowledgements & References

For the completion of this project we partly inspired ourselves by some examples of already provided notebooks on Kaggle.

[1] Link of the competition :
https://www.kaggle.com/competitions/happy-whale-and-dolphin
[2] pretrained models:
https://pytorch.org/vision/stable/models.html
[3] Link of the cropped YOLOv5 dataset:
https://www.kaggle.com/code/awsaf49/happywhale-cropped-dataset-yolov5
[4] ArcFace :
https://paperswithcode.com/paper/arcface-additive-angular-margin-loss-for-deep
[5] Efficient Net:
https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html