Happywhale - Whale and Dolphin Identification

Matthew Harding [Student ID - MHARD94]

April 2024

1 Abstract

A convolutional neural network was created to classify images for individuals from numerous species of whales and dolphins. However, poor performance was achieved in the classification task. It was found that adjustments to epochs, learning rate, batchsize and network architecture lead to little performance gain. It was hypothesised that for this task, fixes to class imbalance and additional image preprocessing would be required in improve model performance.

The code for this project can be found here.

2 Introduction

This project explores the problem of classifiying images of dolphins and whales to identify indivduals. This work is based on the *Happywhale - Whale and Dolphin Identification* Kaggle competition [3].

Data for this competition contains over 50,000 images of over 15,000 unique individual marine mammals from 30 different species collected from 28 different research organizations. Individuals have been manually identified and given an individual_id by marine researchers.

The Kaggle competion had the additional complexity of requiring the ability to classify individuals not included in the training dataset. It was decided to ommit this requirement due to time limitations. The Kaggle competion also measured model performance via Mean Average Precision @ 5 but for this task it was replaced with Loss and Accuracy.

3 Training Data

Looking at the histogram of species Figure 1, there is a clear inbalance in the number of images per species with the vast majority of labelled images being for bottlenose dolphins, beluga whales, humpback whales and blue whales. For some species the number of examples it so low it will be difficult to gain a high level of accuracy.

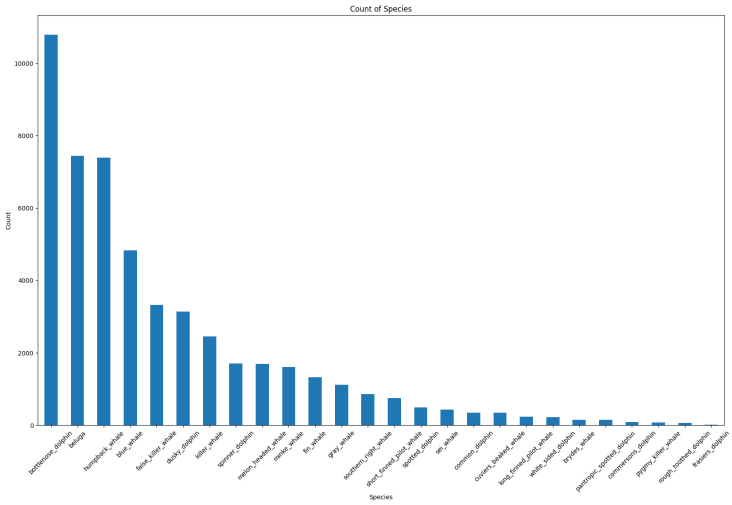

Figure 1: Number of training images per species

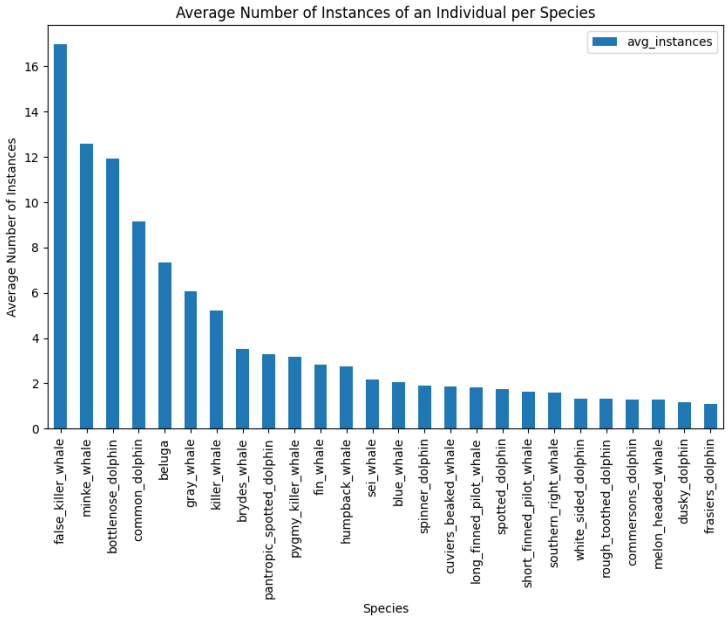

Figure 2: Mean count of images per individual by species

Figure 2 shows the mean count of images per indivdual by species where we see another imbalance with some species on average containing a large number of images per indivudal whilst others only contain one or two images per individual.

Figure 3: Example image of a bottlenose dolphin

Figure 4: Animal detection using YoloV5 model

4 Image Preprocessing

The training images came in a variety of sizes. Figure 3 shows an example unprocessed image. Large amounts of each image are just of water which is irrelvant to the classification task. We are only interested in the parts of the images that contain the animals.

A solution to this issue was proposed within the discussions of the Kaggle competition [1]. YoloV5 was trained on a dataset of 1200 pictures of whale flukes and the corresponding location of points on the edge of the fluke for those pictures [6]. This was then assumed to generalise to finding the bounding box for any part of a whale or dolphin within the water. However, it is worth noting that finding the bounding box of any part of the animal is considered an Out of Distribution problem. From this technique a cropped dataset was created with all images having a dimension of 256 x 256. This is the dataset

that was then used to train the whale and dolphin classifier.

5 Image Transformation

A custom PyTorch dataset class called *WhaleDataset* was created for loading in the training images. One-hot encoding was used for the image lables, representing the individual ids provided.

The images were converted to RGB format as this was the compatible format for PyTorch. The images were then transformed using the TorchVision transforms package. The images were already all the same dimension from the preprocessing outlined in the previous section. The transformation involved converting the PIL image to a PyTorch tensor, scaling ther pixel values to the range [0,1] followed by normalizing the tensor by subtracting the mean and dividing by the standard deviation.

Scikit Learn's *train_test_split* was used to divide the data into training and test datasets using an 80/20 split. A random state was specified for reproducibility between training runs. Shuffling was enabled on the training set.

6 Network Architecture

A convolutional network architecture similar to Figure 5 was selected for this problem. Three convolutional layers were selected for the network architecture so that increasingly more complex and abstract features could be extracted from the input image.

- `conv1` takes the input image with 3 color channels (RGB) and applies 32 different 3x3 convolutional filters to extract low-level features like edges, corners, and simple patterns
- `conv2` takes the output of the previous layer (32 feature maps) and applies 64 different 3x3 convolutional filters to detect slightly more complex patterns and features.
- `conv3` takes the output of the previous layer (64 feature maps) and applies 128 different 3x3 convolutional filters to detect even more complex patterns and higher-level features that are useful for distinguishing between different classes.

After each convolutional layer, a ReLU activation function is applied to introduce non-linearity to the network

After the convolutional layers, there are two fully connected layers. The first fully connected layer takes the flattened output of the convolutional layers and applies a linear transformation, mapping to a fixed-sized vector of 512. This layer acts as a non-linear feature extractor, learning a higher-level representation of the convolutional features. A ReLU activation function is applied after this layer to introduce non-linearity. The final fully connected layer applies a linear

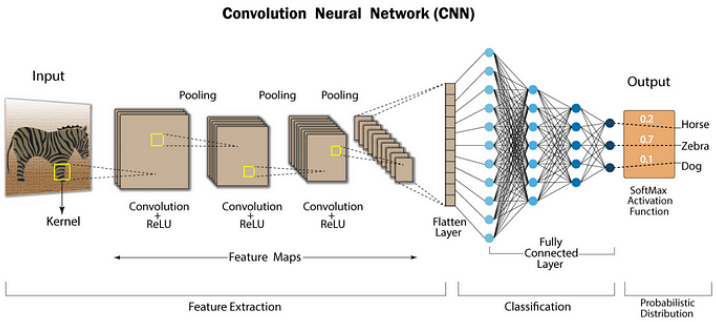

Figure 5: Example of a CNN

transformation, mapping the 512-dimensional vector to a vector of size equal to tthe number of unique individual_ids in the training data.

7 Model Training

Epochs	**Learning Rate**	**Batch Size**	**Loss**	**Accuracy**	**Notes**
5	0.01	32	10.6497	0.87%	
5	0.1	32	10.7591	0.87%	
5	0.001	32	17.1580	0.72%	
5	0.01	16	10.6052	0.87%	
10	0.01	16	10.3002	0.86%	
10	0.01	16	10.0023	0.86%	Added Dropout
10	0.01	16	10.0001	0.86%	Additional Conv2d layer

Table 1: Test Loss and Accuracy with different hyperparameter configurations

The Adam optimizer, a robust gradient-based optimization method, was chosen. Adam allows for faster convergence, compared to stochastic gradient descent, by adapting the learing rate during training [4].

The PyTorch *CrossEntropyLoss* criterion [2] was used as this was a multi-class classification problem. This criterion first applies a Softmax function on the logits, converting them into a propability distribution over the classes. Then the cross-entropy loss measures the performance of the model by comparing the predicted probability distribution with the true distribution (represented by the one-hot encoded target labels). It calculates the negative log-likelihood of the

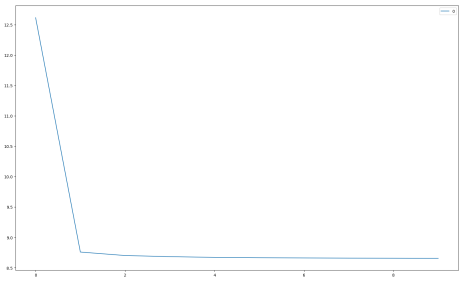

Figure 6: Training loss

true class probabilities, penalizing the model more for being confident about the wrong predictions.

Despite the massive reduction in file size that came from using the cropped dataset instead of the original dataset, training times on the local machine were still prohibitively long. Therefore, training was done using AWS Sagemaker using a ml.p3.8xlarge GPU instance.

The hyperparameters under consideration were

- `Epochs`: Number of complete passes of the training dataset through the network
- `Learning Rate`: The size of the step taken adjusting the model parameters during the optimization stage
- `Batch Size`: The number of training examples passed through the network before backpropagation and optimization

An intial training run using a batch size of 32, a learning rate of 0.01 and training for 5 epochs gave a test accuracy of 0.87%. Given the number of classes, this translates to the model predicting correctly 8 or 9 out of 1000 samples. Increasing the learning rate had no impact on performance whilst reducing the learning rate reduced the performance. A smaller learning rate should lead to a higher accuracy [5] so this may be due to underfitting where the model parameters doesn't receive sufficient updates.

Reducing batch size made no material difference to performance. Smaller batch-size reduces the risk of overfitting however the model may already be well-regularized.

Training for additional epochs lead to a small improvement in accuracy. However, looking at Figure 6, the training loss was beginning to converge beyond the 5th epoch so continued training risked overfitting.

Dropout was added to the classifer architecture with a dropout rate of 0.2. Again this made no difference to performance so again overfitting is not the cause of poor performance.

A forth convolutional layer was then added to the network structure under the assumption a deeper network could capture more patterns within the images but also did not yield performance improvements.

8 Conclusions

Performance achived by the model can be considered poor. This can be attributed to several factors

- Differences between individuals with a species are very slight and can change over time
- Large class inbalance in the number of examples per species and per individual
- Computational limitations meant each model could only be trained for a few epochs

With additional time and computational resources, it is beleived performance gains could be made by first creating a model to classifiying the image for species before passing on species specific classifier that could then classify for the individual. The assumption being that a species specific model would be easier to train to capture the nuances of each individual.

Additionally, it would be of interest to see if performance improvements could be made via contrast enhancement as an image preprocessing step. The animals were similar colors to the ocean so something that could emphasis the shape of the animal could assist the model.

References

[1] Happywhale: Cropped data prepare yolov5. `https://www.kaggle.com/code/awsaf49/happywhale-cropped-data-prepare-yolov5`.

[2] Pytorch cross entropy loss. `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`.

[3] Happywhale - whale and dolphin identification. `https://www.kaggle.com/competitions/happy-whale-and-dolphin`.

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

[5] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. Don't decay the learning rate, increase the batch size, 2018.

[6] Whale fluke location labelled dataset. `https://www.kaggle.com/datasets/martinpiotte/humpback-whale-identification-fluke-location`.