University of Osnabrueck

Implementing ANNs with TensorFlow

Turtle Recognition Using Deep Convolutional Neural Networks

Dennis Hesenkamp, Lennart Zastrow, Madhuri Ramesh

supervised by
Charlotte Lange

April 10, 2022

Contents

Abstract

Sea turtles are endangered but very important for the ecosystems they live in. Being able to track and identify them is critical for modern conservation. Their unique head patterns allow for identification from photographs with neural networks. As part of a machine learning competition on the platform Zindi [1], we built a convolutional neural network to identify sea turtles and investigate how different, commonly use image augmentation techniques influence the model's ability to learn the features of the turtle images. The accompanying code can be found in this repository: `https://github.com/dhesenkamp/turtleRecall`

1 Introduction

Three-fifths of all turtle species worldwide are on the verge of extinction or are already severely threatened (Goldstein, Goldstein, Goldstein, & Goldstein, 2021; Mortimer & Donnelly, 2008; Paladino & Morreale, 2001). This makes turtles the most endangered vertebrates in the world, ahead of mammals, birds, fish, and amphibians.

Sea turtles are known as indicator species which means that their presence and abundance reflect the health of the wider ecosystem. Therefore, increasing our ability to identify and understand them can enhance our ecological understanding. The herbivores, carnivores, or omnivores are at the same time hunters, pest controllers, and food sources for other animals. The scavenging species, for example, ensure a clean environment, and the herbivorous turtles make an important contribution to spreading plant seeds. They maintain beds of seagrass and coral reefs and facilitate a nutrient cycle from marine to land ecosystems. Humans are making it difficult for the turtles to survive. The animals are suffering from climate change, habitat destruction, excessive trade in live animals, the sale of their meat and shells, and environmental pollution.

Deepmind [2], Ocean Hub Africa [3], and Local Ocean Conservation [4] have come together to provide a dataset of sea turtle images and a challenge to built a machine learning model for identification of sea turtles by their facial scales. We use this dataset to built a convolutional neural network (CNN) that is supposed to do exactly that: recognise known turtles via their unique head patterns from images or classify them as new, so far unknown individuals.

2 Grant Species Preservation Through Population Census

The ability to distinguish between individuals of the same species is a fundamental tool for modern animal welfare. To ensure the protection of individuals, it is crucial to identify their whereabouts and movement patterns. Because sea turtles are a powerful indicator of overall ecosystem health, accurate identification serves to enhance our ecological understanding.

2.1 Turtle Tagging

In the past, the detection and tracking of individuals was realised by attaching tags to the fins of the individuals found. This method is severely compromised by the loss of tags and thus the successful tracking of population dynamics is not guaranteed. Basically, the use of *flipper tags* was common, which are mostly made of metal and plastic. This method is very costly due to the extraordinarily long life span of the turtles, which means that the flipper tags were lost or identification was no longer possible due to wear and tear of the material.

[1] `https://zindi.africa/competitions/turtle-recall-conservation-challenge`
[2] `https://deepmind.com/`
[3] `https://ocean-innovation.africa/`
[4] `http://localocean.co/`

2.2 Photographic Identification

Just as the human finger print, turtles have unique and time-stable facial scales by which they can be identified (Carpentier, Jean, Barret, Chassagneux, & Ciccione, 2016). Photographic identification has become the method of choice over time as it is non-invasive and low-cost. Due to the advances in machine learning and object identification, we are able to identify turtle individuals with algorithms as opposed to humans manually searching through a database of images. The goal of such machine learning algorithms is to assign a unique ID to each individual if it already exists in the database and to create a new ID if a new individual has been sighted.

3 Convolutional Neural Networks

In the field of machine learning, especially in the deep learning sector, a CNN is a deep learning algorithm. It can take an input image and reproduce an identification of the object with a certain probability. The special feature of a CNN is that it is able to learn filters of different types (e.g. horizontal lines, vertical lines, etc.). A convolutional neural network has a characteristic structure in terms of its layers. It is structured in convolutional, pooling, and fully connected layers. The arrangement of alternating convolutional and pooling layers allows a more accurate and complex analysis of the image. The first layers focus on shapes and colors, while later layers contribute to the identification of more complex details for the recognition of the overall image.

3.1 Convolutional Layer

The convolutional layer is the key component of a CNN. It contains a certain set of filters, also called kernels. The parameters of the filter are learned over the course of the training. The filter interacts with the image and convolves it. From this convolution, an activation map is created which is calculated from the dot product between each element of the filter and the input. The weights in the filter are maintained as the filter moves across the image. However, these weights adjust during backpropagation and the associated gradient descent to achieve the most accurate results.

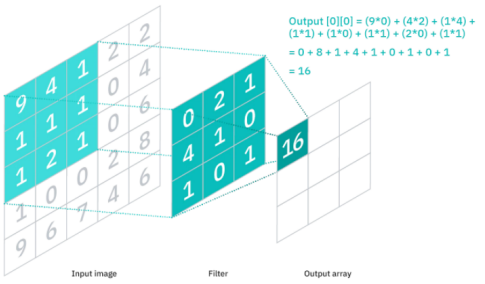

Figure 1: Convolutional Layer

3.2 Pooling Layer

Since a great increase in dimension occurs through the use of a convolutional layer, a dimension reduction is required in the next step to reduce the number of parameters of a CNN. This is achieved by the pooling layer. It has the advantage that the computational cost decreases

drastically. Also, unnecessary details are omitted, which is helpful for the later image identification.The most commonly used pooling methods are maximum and average pooling. In figure 2, you can see how a maximum pooling is performed, leading to a dimension reduction.

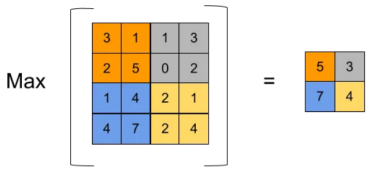

Figure 2: Max Pooling Layer

3.3 Fully-Connected Layer

Since the individual pixel values of the input image are not directly connected to the output layer, a fully-connected layer is required which is directly connected to the output layer. The layer does the classification using the collected features from the previous layers. At the end, a softmax or Sigmoid function (dependent on the problem) is applied, which outputs a classification using probabilities between 0 and 1.

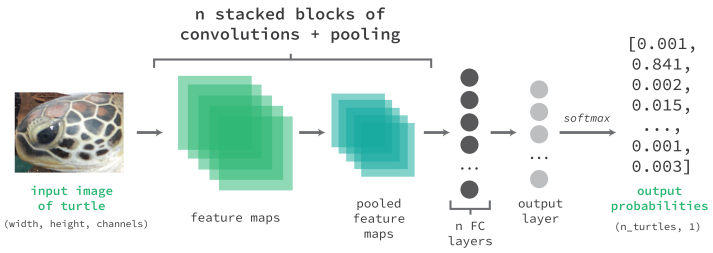

Figure 3: Turtle Image using CNN

4 Data

The dataset is provided with the *Turtle Recall: Conservation Challenge*[5] and consists of images of turtle faces. The images are labelled, each turtle has a unique ID. Besides the angle from which the image was shot (top, right, left) and the ID, no further info is provided about the images.

The images themselves have three colour channels, come in different sizes, and many have timestamps or handwritten identification tags in the background. They are split in three parts: a set of images for training, one for testing, and a large set of extra images.

The training set consists of 2145 images from 100 unique turtles. The set with the extra images comes with an additional almost 11000 images from 2231 different turtles, some already contained in the train set. This yields a total of about 13000 images and 2265 turtles. The test set only comes with images but without the annotation of turtle IDs, as it is meant to be used for model evaluation in the priced competition. However, this means that the test images are without use for our purpose.

[5] `https://zindi.africa/competitions/turtle-recall-conservation-challenge/data`

Because the training set is rather small, we also make use of the extra images. A quick exploratory analysis shows that the dataset is both hugely unbalanced and that there are only less than 6 images per turtle on average, with a median number of 3 images per turtle. Because such a small amount of data per class will likely lead to very poor approximation results, we decide not to use the entire available data.

4.1 Preprocessing

Before actually using the images and feeding them into a neural net, we need to perform some necessary as well as some optional preprocessing steps in order to bring the image data into a more helpful format.

As a first step, we get rid of any turtles with less than a specified number of images in the dataset. We require each turtle to be represented by at least 10 images in the dataset. This reduces the total number of turtles to 253 and the total number of images to roughly 5000. Turtles now have an average of 20 images and a median of 14 images. There is still a considerable imbalance in the data, as can be seen in figure 4.

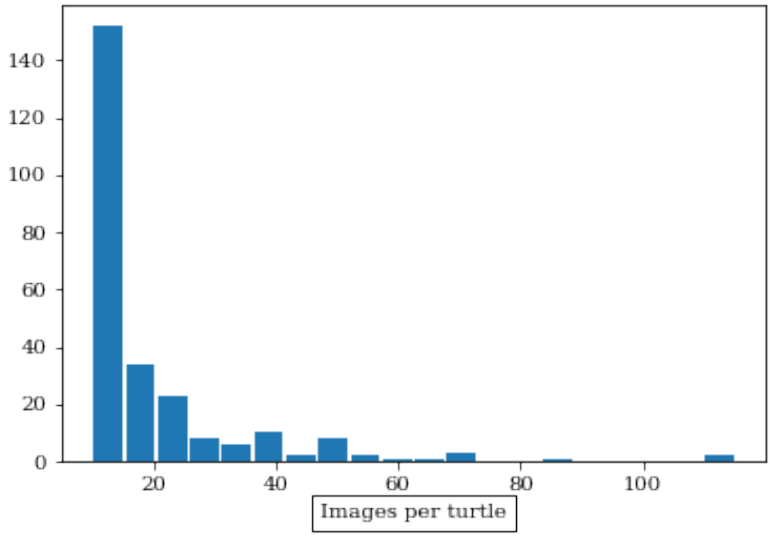

Figure 4: Number of images per turtle. The left skew of the distribution means there are many images with the minimum amount of images, while only a few have considerably more.

We further crop and resize the images such that they are all in the same shape and size, namely 224×224 pixels. Before doing any computations on the image data, we also convert them to numerical arrays, yielding us a three-dimensional array of size $224 \times 224 \times 3$ for each used image. Since the colours are in the RGB colour space, the values are now in the range 0 to 255, which we normalise to a range of 0 to 1. This has been shown to increase training speed of deep networks (Ioffe & Szegedy, 2015). We further apply a one-hot encoding to the classes, as the alpha-numerical strings the turtle IDs are supplied in are rather meaningless to a neural net. At this point, we are technically ready to feed the images into a network for training.

4.2 Data Augmentation

The dataset, unfortunately, contains very little data unevenly distributed among the classes. A low number of training examples per class can lead to the class not being learned efficiently

and may even impede the entire training process (Huh, Agrawal, & Efros, 2016). To avoid this, we apply a set of basic augmentation techniques to multiply the data available for training:

Image rotation We rotate the images by 90, 180, and 270 degrees. This immediately quadruples the available data. The basic characteristics of the pattern on the head of the turtle are preserved.

Gaussian filter We also apply a Gaussian filter to introduce a blur and reduce detail in the image. Mathematically, this is the same as convolving the images with a Gaussian function.

Random HSV The HSV colour space uses hue, saturation, and value to describe colours, whereas the RGB colour model uses a linear combination of the colours red, green, and blue to describe the same. We transform the RGB values to HSV, randomly modify them within a specified range, and convert back to RGB.

Additive noise Lastly, we add random numerical values, sampled from a normal distribution with very low standard deviation, to the existing image values, followed by clipping the values within range 0 to 1 (our normalisation range).

Applying the above techniques, we have enhanced our dataset and also somewhat shifted the distribution of images per turtle to be more desirable. Further common augmentation techniques include random brightness shifts, grey level mapping, histogram equalisation, image shifting and shearing. Example images of the used augmentations can be seen in appendix A.

5 Models

Different models and model architectures are the result of (restricted) computational resources, new ideas and algorithms, specific tasks, and so on. To approach our problem, we figured that experimenting with different models would likely point us into a direction of certain architectural features which work well with the data at hand.

5.1 AlexNet

Krizhevsky, Sutskever, and Hinton have presented their influential *AlexNet* in 2012. It consists of only five convolutional layers, some additional max-pooling layers and three fully connected layers at the end, so its structure is rather simple. Nonetheless, it is packed with more than 40 million trainable parameters (the fully connected layers at the end are very large), making it not exactly computationally cheap to train. An AlexNet was supposed to serve as our baseline model from which we wanted to build things up.

5.2 EfficientNet

We then implemented a pre-trained EfficientNet with custom top layer. The EfficientNet model family was first introduced in 2019 (Tan and Le) with the idea of providing scalable CNN architectures. Their architecture is largely based on MobileNets inverted residual blocks, in which bottlenecks serve as in- and output of the residual connections (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018). Multiple networks of different size for image classification purposes were introduced, many achieving better results than state-of-the-art models like the MobileNet it built upon and ResNet while being significantly smaller in terms of depth, width, and image resolution (Tan & Le, 2019). Two years later, a second, even more powerful

and efficient generation was introduced: EfficientNetV2 (Tan & Le, 2021). For our purposes, given the low input resolution of 224×224, the model of choice is the EfficientNetV2-B0 – the smallest model of the ensemble. The model is pre-trained on ImageNet21k and we implemented a custom fully-connected layer at the end to match our classification task.

5.3 InceptionNet

For an additional comparison, we implemented yet another supposedly very efficiently designed CNN, the InceptionV3 as introduced by Szegedy, Vanhoucke, Ioffe, Shlens, and Wojna in 2015. Its computational low cost makes the Inception architecture an attractive choice when resources are limited, e.g. in mobile scenarios. The inception network was a milestone in the development of CNN classifiers. It achieved extremely high accuracies on the ImageNet dataset with InceptionV3 reaching 78.1% accuracy. The fundamental idea behind the inception network is the inception block. In a traditional convolutional neural network layer, the previous layer's output is the input of the next layer until the state of prediction is accomplished. The inception block takes apart the individual layers. The previous layer's output is passed to four different operations in parallel and their outputs is concatenated – the network is made wider instead of deeper. The naïve approach consists of a 1×1 convolution layer, a 3×3 convolution layer and a 5×5 convolution layer followed by a maximum pooling operation and a concatenation layer. Due to the high computational cost especially of the 5×5 filter, the 1×1 filter is first added to the naïve inception module. This leads to a reduction in computations of 90%. Additionally, the 1×1 convolutional filters allow learning cross-channel patterns across the depth of the input data.

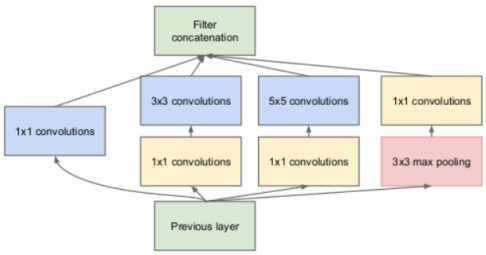

Figure 5: Inception Block

6 Evaluation

When finally wanting to run and evaluate the models, we encountered problems with the execution of our pipeline. It turned out that the training process requires dramatically more computational resources than we anticipated - more, at least, than our local machines could handle. We tried different approaches to cope with this problem: As a first step, we skipped the AlexNet as baseline model. It has a huge number of trainable parameters compared to the pre-trained EfficientNetV2 (small amount of trainable parameters) and InceptionV3 (somewhat in the middle between the former two), requiring much more backpropagation computations within the training. Unfortunately, just having fewer trainable parameters did not reduce the computational load enough for us to be able to train the models.

The next logical consequence for us was to reduce the size of the dataset. We initially augmented the data, increasing the size of the dataset roughly 10-fold, to enable the models to better learn features of the turtle head patterns and avoid overfitting. Step by step, we cut out the different augmentations until we were back to the original, non-augmented dataset.

	Trainable parameters	Total parameters
AlexNet	47,790,398	47,793,150
EfficientNetV2-B0	6,184,078	6,244,686
InceptionV3	22,288,798	22,323,230

Table 1: Parameters per model with 254 classes. Number of classes only insignificantly affects total number of parameters.

Even this dramatic decrease of dataset size back to around 5000 images did still not allow us to train the models properly. We then went on to experiment with different buffer sizes for the shuffle procedures and different batch sizes to reduce the load on the RAM. Although decreasing both of these parameters seemed to free up RAM, a full training was still not possible. The EfficientNetV2, which was for now the model on which we tried to train, was able to run the fitting for 1-3 epochs before sessions crashed due to full RAM. Even further reducing the dataset size by increasing the number of minimum images per turtle from 10 up to 15, which lead to a further reduction of the total number of images to little over 3000, did still not allow for successful training.

CNNs can generally be scaled along three dimensions: changing the depth of the network, i.e. the number of layers, the width of the network, e.g. by increasing the number of channels per convolutional layer, or by changing the resolution of the input images. Because we did not want to make major changes to the layer architecture of the models we use, we decided that downsampling the images would be the next measure to explore. The resolution we used so far yields $224 \times 224 \times 3 = 150528$ values per image, we reduced to 160×160 pixels with a resulting decrease of ~50% with regard to image values: $160 \times 160 \times 3 = 76.800$. Reducing the dataset size (although not the number of samples) again by half with this approach, we managed to successfully train the EfficientNetV2. We were not happy with this approach since reducing the image size potentially gives rise to unrecoverable loss of important information (Zhang, Zhao, Zhang, Xiong, & Gao, 2011). Further, both the EfficientNetV2 and the InceptionV3 expect input sizes of 224×224.

Because of this, we turned to Google Colab to execute our pipeline. The resources there allowed us to return to the previous resolution of 224×224 pixels. Treating the minimum number of images per turtle (which directly affects the number of samples in the dataset) as a first hyperparameter to explore - we still could not run the training with the entire augmentation pipeline due to RAM restrictions - we found a dataset size of approximately 5000 images to max out on Colab's freely available resources. A table with how the minimum number of images affects the dataset size and median number of images per turtle can be found in table 3 in appendix B. Further naïvely exploring the batch size, we found batches of 64 samples to best work with the computational resources while still allowing for a feasible training speed. The convergence behaviour did not seem to be significantly affected by different batch sizes.

A first run with both the EfficientNetV2 and InceptionV3 for 10 epochs to establish a baseline yielded the results shown in figure 6.

The EfficientNetV2 is able to achieve accuracy 97.02% on the training set while the InceptionV3 - pre-trained on the same ImageNet21K - achieves 90.18%. Validation accuracies are 49.83% and 17.01%, respectively, and test accuracies are 50.42% and 16.25%, suggesting that the InceptionV3 actually overfits on the training data. Although no definite convergence behaviour could be seen at this point, we decided to focus on the EfficientNetV2 – mainly due to the limited computational resources. We can, however, not draw any meaningful conclusion regarding the InceptionV3 since we did not explore this direction any further.

After establishing this baseline using only non-augmented images, we went on to assess to

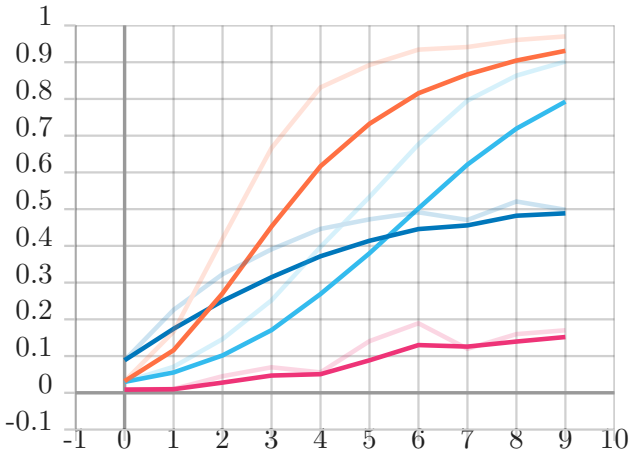

Figure 6: Categorical accuracy on training and validation data for EfficientNetV2 (train: orange, validation: blue) and InceptionV3 (train: light blue, validation: pink).

what extent the different augmentations influence model performance. We first reduced the size of the dataset by increasing the minimum number of images per turtle to 13 (from 10), and then augmented a part of this data using only a single augmentation technique to get the image number up to roughly 5000 again. The comparison between baseline and augmentations, which can be seen in figure 2, has to seen skeptically, though, because increasing the number of required images per turtle does a) significantly reduce the number of classes (unique turtles) from 254 to 150 and b) provide more total data per class, which theoretically should allow the model to better capture the underlying distribution.

	Train	**Validation**	**Test**
Baseline	97.02%	49.83%	50.42%
180° rotation	93.14%	89.38%	58.98%
Gauss-filter	96.76%	96.72%	63.80%
Random HSV	98.67%	98.46%	56.77%
Noise	92.27%	86.15%	56.77%

Table 2: Categorical accuracy of EfficientNetV2 on differently augmented datasets after 10 epochs.

As can be seen, validation and test accuracies greatly benefit from all augmentations, with the Gaussian filtering considerably outperforming the other techniques on the test data.

7 Conclusion

Due to the limitations faced within execution of our project, we can, unfortunately, only draw interim conclusions here. The AlexNet could not be tested by us, but of the EfficientNetV2-B0 and the InceptionV3, the former one appears to be the more promising model for recognising facial scales of turtles. The pre-trained EfficientNet model family has been shown to transfer well to other problems (Tan & Le, 2019). This can most likely be attributed to the very effective scaling method employed by EfficientNets, which allows the model to focus more on the relevant, detail-rich regions within an image (Tan & Le, 2019).

8 Outlook

We consider achieving a test accuracy of 63.80% given the size and characteristics of the dataset after 10 epochs of training already a great success, but we could not evaluate the models for their full potential within the scope of this project. With more computational resources available, the next obvious step would be to increase the size of the dataset to include all augmentations while also including more turtles again by lowering the number of minimum images per turtle. Then, it would also be interesting to conduct a more thorough hyperparameter search, which was not feasible for us. For example, we only used Adam with standard parameters as optimizer because it is computationally efficient and requires little memory (Kingma & Ba, 2014). The EfficientNet model family has been trained with Hinton's unpublished RMSProp (Tan & Le, 2019). Further, we only reported categorical accuracy as model evaluation metric. In the original Zindi competition, mean average precision at top 5 (MAP@5) has been suggested for this problem, probably because of the many classes to consider. Another interesting approach could be a less aggressive downsampling of the images. As proposed in the original competition, we scaled the images down to a resolution of 224×224, exactly the resolution that is also suggested as input for the EfficientNetV2-B0. As already discussed earlier, EfficientNet is a model family and B0 the smallest one in the ensemble. By scaling up the image resolution, it would not only be possible to include more information in the images themselves, but also to employ a larger and stronger model.

Further, we manually designed the augmentation. Doing so effectively is usually dataset-specific and requires expert knowledge in the image domain, as different techniques work best for different domains. Cubuk, Zoph, Mane, Vasudevan, and Le have proposed their AutoAugment image augmentation procedure in 2018, in which a search space of augmentations is explored by a controller RNN to find the best procedures given the data at hand.

The competition on Zindi is not yet over at the time of writing this documentation, but the top-5 leaderboard scores already achieve a MAP@5 of more than 98%. We are excited to see which procedure ultimately works best on this problem.

References

Carpentier, A. S., Jean, C., Barret, M., Chassagneux, A., & Ciccione, S. (2016). Stability of facial scale patterns on green sea turtles chelonia mydas over time: A validation for the use of a photo-identification method. *Journal of Experimental Marine Biology and Ecology*, *476*, 15-21. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0022098115300733` doi: https://doi.org/10.1016/j.jembe.2015.12.003

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2018, 5). Autoaugment: Learning augmentation policies from data. *Cvpr 2019*, 113-123. Retrieved from `https://arxiv.org/abs/1805.09501v3` doi: 10.48550/arxiv.1805.09501

Goldstein, J. P., Goldstein, E. R., Goldstein, B. I., & Goldstein, M. I. (2021, 1). Imperiled sea turtles: Ecology and conservation. *Reference Module in Earth Systems and Environmental Sciences*. doi: 10.1016/B978-0-12-821139-7.00062-3

Huh, M., Agrawal, P., & Efros, A. A. (2016, 8). What makes imagenet good for transfer learning? *CoRR*. Retrieved from `https://arxiv.org/abs/1608.08614v2`

Ioffe, S., & Szegedy, C. (2015, 2). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*. Retrieved from `http://arxiv.org/abs/1502.03167`

Kingma, D. P., & Ba, J. L. (2014, 12). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track*

Proceedings. Retrieved from https://arxiv.org/abs/1412.6980v9 doi: 10.48550/arxiv.1412.6980

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks.. Retrieved from https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdfhttp://www.cs.toronto.edu/~kriz/imagenet_classification_with_deep_convolutional.pdf

Mortimer, J. A., & Donnelly, M. (2008). *Eretmochelys imbricata. the iucn red list of threatened species 2008: e.t8005a12881238.* Retrieved from https://dx.doi.org/10.2305/IUCN.UK.2008.RLTS.T8005A12881238.en

Paladino, F. V., & Morreale, S. J. (2001, 1). Sea turtles. *Encyclopedia of Ocean Sciences: Second Edition*, 212-219. doi: 10.1016/B978-012374473-9.00443-4

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018, 1). Mobilenetv2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 4510-4520. Retrieved from https://arxiv.org/abs/1801.04381v4 doi: 10.48550/arxiv.1801.04381

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015, 12). Rethinking the inception architecture for computer vision. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-December*, 2818-2826. Retrieved from https://arxiv.org/abs/1512.00567v3 doi: 10.48550/arxiv.1512.00567

Tan, M., & Le, Q. V. (2019, 5). Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*.

Tan, M., & Le, Q. V. (2021). Efficientnetv2: Smaller models and faster training. *CoRR*, *abs/2104.00298*. Retrieved from https://arxiv.org/abs/2104.00298

Zhang, Y., Zhao, D., Zhang, J., Xiong, R., & Gao, W. (2011, 11). Interpolation-dependent image downsampling. *IEEE Transactions on Image Processing*, *20*, 3291-3296. doi: 10.1109/TIP.2011.2158226

Appendices

A Augmented Images

Figure 7: Non-augmented image.

Figure 8: Image rotated by 180 degrees.

Figure 9: Image with Gauss-filter applied.

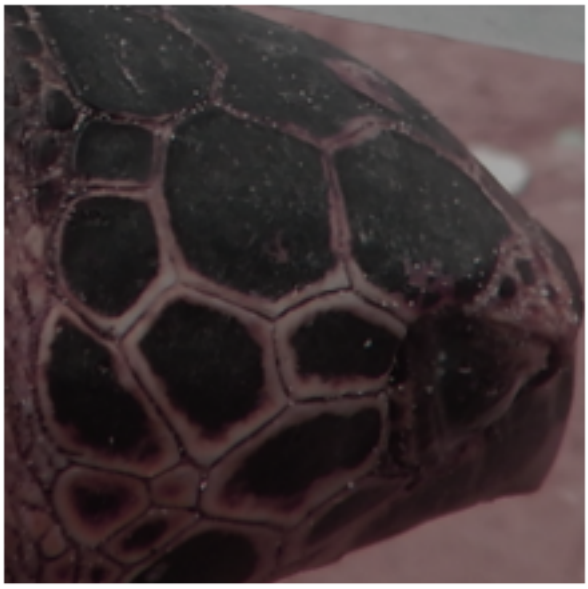

Figure 10: Image with randomly changed HSV values.

Figure 11: Image with added noise.

B Dataset

Min number images	8	9	10	11	12	13	14	15
Images in dataset	5846	5638	4990	4690	4338	3846	3560	3392
Images per turtle (median)	12	12	14	15	16	18	21	22

Table 3: Dataset size in relation to number of images per turtle.