Automatically classifying clownfish pictures datasets at the species level

Léonard Jequier

CBG
University of Lausanne

December, 2018

Abstract: Amphiprion **species, commonly called clownfishes, are a valuable evolutionary model. Researchers are interested in computationally studying morphological variation between clownfishes species in order link theses variations to their molecular bases. To achieve this, they need as much data as possible, consisting of pictures of clownfish specimen labeled with the species name. Currently, labelling the picture is done manually and require a lot of time and expertise. The aim of this project is to create a program capable of providing species label to a sets of clownfish pictures. To do so, the best currently available image recognition technology was used: convolutional neural networks. Among the created programs, one can distinguish between pictures of *A. clarkii* and *A. perideraion* with an accuracy of 96%. Another can distinguish *A. ocellaris* and *A. frenatus* in addition to the two species mentioned before with an accuracy of 89%.**

Keywords: *Amphiprion*, Image recognition, Convolutional neural network

I. Introduction

Amphiprion spp. is a valuable evolutionary model. It is a rare example of adaptive radiation in marine environment[2] and they also form a famous and very interesting mutualistic interaction with anemone. Several groups including Nicolas Samamin's lab are work on molecular and phenotypic evolution of the different species of that family. To computationally analyze evolution of morphology of *Amphiprion spp*, they need large datasets of clownfish images sorted by species. As the species is quite popular, images of clownfish are largely available online but manually classifying them takes time and expertise. In that context, having a program able to rapidly and accurately classify the images would be very beneficial. That is toward this goal that this first step project was aiming.

This is an image classification problem: the program should take an input image (the picture of a clownfish) and predict its class (its species). The best method presently available for that purpose is Convolutional Neural Networks (CNN). Indeed, the best existing image recognition program are based on this technique like the winner of the 2017 Imagnet image recognition contest [6]. In their most simple form, CNNs are typically made of two parts, the first contains convolutional layers. They consist of filters detecting more and more complex pattern in the image and reducing the input dimensionality. The second part contains so called "fully connected" layers, which classify the input images based on how these complex pattern correlates with the different classes. The

pattern the network will be looking for and the weight activation of these patterns by an image has in its classification are not supervised by the creator of the program. Instead, they are randomly initialized, and the network is trained on a labeled dataset. The training process takes place in two steps. The first is called feedforward and consists of inputting the training dataset to the network and compare the prediction it makes to the known class of the images. This comparison is called the loss function and measure how badly the network is classifying the input pictures. The second step is called backpropagation and consists of updating the weights and filters in order to minimize the loss function and this in generally done using stochastic gradient descent. After that several optimization cycle on the training dataset (one cycle is called an epoch), new images can be input to the network. The network will look for the pattern in the images and based on that, output the most probable class for the image.

An approach that can be used to create a CNN is to build the network architecture from scratch, randomly initialize the weights and filters and train it on the dataset. But one can also use a network that was already pre-trained to classify another dataset and train it again on the dataset of interest. This approach also works because the pattern detected by the layers are very simple and will be similar no matter the dataset the network is trained on. The second approach is particularly useful when the datasets are small.

The aim of this project was to use python and a package called Pytorch to create a CNN that take as an input pictures of clownfish and outputs a predicted species name for each image. Milestones of the project were: gather the database; organize the pictures in a form usable by the program; chose and built the neural network architecture; load, normalize and apply data augmentation steps to the images, train the network on the SCITAS cluster and finally analyze the predictions.

II. Results and discussion

a. Gathered dataset

The complete dataset contained 28 species and 2220 pictures in total. But the number of images per species is importantly unbalanced and often too low to efficiently train a network. Therefore, I decided to begin by using the two species for which I had the most pictures. After that, a second attempt with four species was done. In both case, 10% of the image of each species were randomly sampled to create a validation dataset. For the training dataset, a trade-off between a low number of image and an unbalanced dataset was necessary. So, the number of images per species was limited to twice the number of images of the least represented species. A summary of the two datasets used is provided in Table 1.

b. Two species classifiers

Two types of CNNs were used. The first was constructed using Pytorch with an architecture adapted from Qin et al and started with random initialization (abbreviated RI2). The second was pre-implemented in pytorch, based on Resnet18 and was pre-trained on the Imagenet dataset. (TL2). Change in loss and accuracy during the training process are shown in Figure 2 and Figure 1. It shows that both network are learning: their accuracy is increasing in the first epochs (An epoch is a complete training cycle on training dataset) and they present no sign of overfitting. This happen when the network becomes so optimized for the training dataset that it starts not being able to classify images outside of that dataset. When this happens, iIt can be detected by looking at the change in loss and accuracy during the training process. If for the training dataset the accuracy is increasing and the loss decreasing but in the same time the opposite happens for the validation dataset, the model is overfitting.

Dataset	Species	Training	Validation
Two species	*Clarkii*	380	59
	Perideraion	228	25
	Total	**608**	**74**
Four species	*Clarkii*	234	59
	Perideraion	228	25
	Ocellaris	211	23
	Frenatus	140	16
	Total	**813**	**123**

Table 1: Number of images per species in the two datasets.

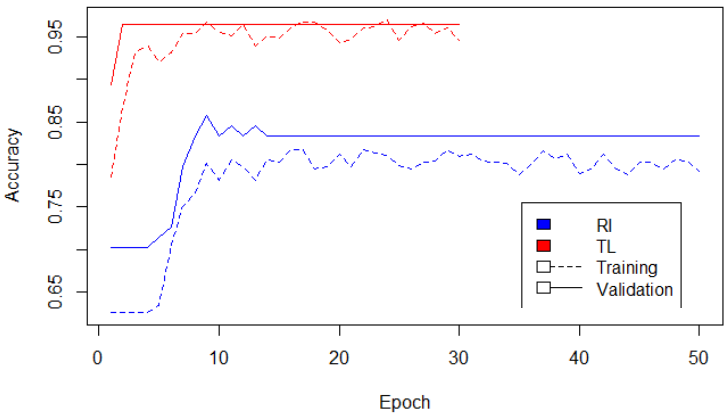

Figure 1: Graphical representation of the change in loss during training for RI2 and TL2.

Model	Global accuracy	Species	Accuracy
RI2	72/84 =0.85	*Clarkii*	57/59 =0.96
		Perideraion	15/25 =0.60
TL2	81/84 =0.96	*Clarkii*	57/59 =0.96
		Perideraion	24/25 =0.96

Table 2: Accuracy after training of RI2 and TL2, on the validation dataset, globally and per species.

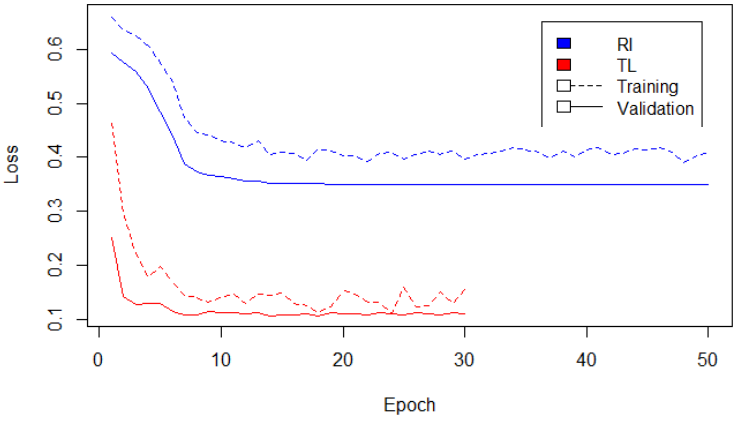

Figure 2: Graphical representation of the change in accuracy during training for RI2 and TL2.

After training, RI2 yielded an accuracy of 85% on the validation dataset and TL2 had an accuracy of 96% (see Table 2). But, with more careful analysis, it appears that the difference in accuracy is mainly due to the fact that RI2 has trouble classifying *A. perideraion*. Indeed, the accuracy for that species is of 0.6, perhaps because the training dataset was smaller, causing a bias in prediction. In TL2 however, this doesn't seem to have an effect as the accuracy in the same between species. It is interesting to note that 2/3 of the false prediction of TL2 are also wrongly predicted in RI2. In the end, for both model the accuracy is higher than expected by chance.

c. Four species classifiers

As the result with two families were promising, both models were adapted to classify four species. The one started with random initialization will be referred as RI4 and the one using transfer learning TL4. The training process for these networks is presented in Figure 4 and Figure 3. Both networks show an increase in accuracy and no sign of overfitting.

RI4 has a global accuracy of 66% and TL4 89% (see Table 3). Accuracy varies between species and is particularly low for *A. ocellaris*. As opposed to RI2 where I hypothesized that the lower accuracy for one species

Model	Global accuracy	Species	Accuracy
RI4	82/123 = 0.66	*Clarkii*	42/59 = 0.71
		Perideraion	18/25 = 0.72
		Ocellaris	11/23 = 0.47
		Frenatus	11/16 = 0.68
TL4	110/123 = 0.89	*Clarkii*	57/59 = 0.96
		Perideraion	23/25 = 0.92
		Ocellaris	16/23 = 0.69
		Frenatus	14/16 = 0.87

Table 3: Accuracy after training of RI4 and TL4 on the validation dataset, globally and per species.

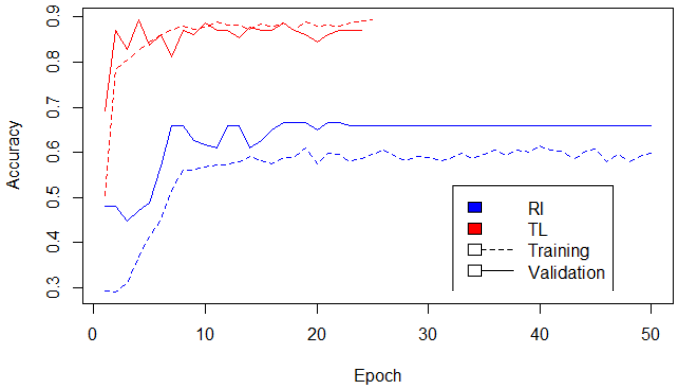

Figure 3: Graphical representation of the change in accuracy during training for RI4 and TL4.

		Predicted:			
		Clarkii	*Perideraion*	*Ocellaris*	*Frenatus*
Labeled:	*Clarkii*	57	0	1	1
	Perideraion	0	23	1	1
	Ocellaris	0	2	16	3
	Frenatus	2	0	0	14

Table 4: Predictions of TL4 compared to the true labels.

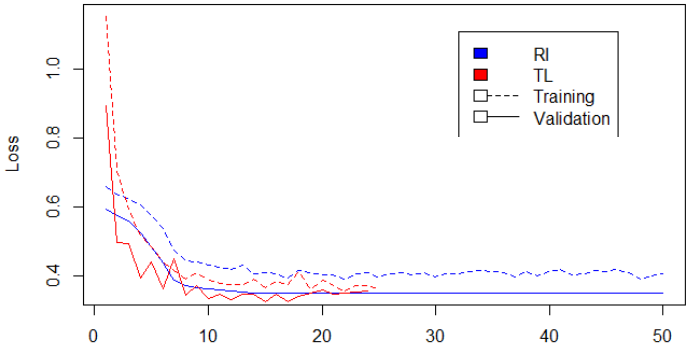

Figure 4: Graphical representation of the change in loss during training for RI4 and TL4.

was due to a smaller dataset, here it is less likely the case, because *A. frenatus* has an even smaller training dataset (210 picture for *A. ocellaris* vs. 140 for *A. frenatus*) but had higher accuracy. To further investigate the wrongly classified images I looked how does the prediction compared to the true labels (Table 4). Before looking at the results, I would have assumed the network would have confounded *A. perideraion* with *A. frenatus* and inversely, because both have a single thin white band near the head. I would also have assumed that it might confuse *A. clarkii* with *A. Ocellaris*. They not as similar at the two above but both have large white bands on their body. But looking at Table 4 doesn't seem to confirm my presupposition. This seem to indicate that some other parameter than the banding pattern might also influence the classification. For example, one could think of the anemone as a potential pattern. Indeed, some clownfish are very specialized to live in only one anemone while other are generalists can live in different species of them [3]. The anemone is present on the background of an important part of the picture of the training dataset. It might have therefore have an influence on the classification process.

d. Four species or "other" classifier

In the form described above, TL4 can classify a dataset only if it doesn't contain outlier (pictures of species not present in its training dataset). That would make its application to classify clownfish images from large image repositories like Flickr difficult. To improve that, I tried to change the architecture and to add a fifth class. The training dataset for that class contains 312 images sampled from all the other *Amphiprion spp* together. This second version will be abbreviated TL4.2 in the text below and its training process is represented in Figure 6 and Figure 7. However, with that modification, the global accuracy drops to 70% (Table 5). Looking at the results for each family, the accuracy in almost all families is lower compared to TL4, except for *A. ocellaris*. The decrease is particularly strong for *A. clarkii*. Moreover, the classification of "other" is particularly inaccurate. I think it is because classification of "other" as a fifth class isn't a good approach to the problem. As said before, neural networks work by looking in the input image for pattern that correlates with each class in the training dataset. But by definition the class "other" has no specific pattern. Would I have the time, I would rather use the fact that the last layer of TL4 outputs a vector of confidence score that the image comes from each of the four classes. Above, we simply took the maximum of that vector and considered it as the predicted class. One could instead only predict as being in a family if the confidence in prediction is above a given threshold and classify as other if it is below.

III. Conclusion

The accuracy of the networks created remain a low compared to other CNNs. DeepFish by Qin et al for example reach an accuracy of 98.57% for classification of 28 fish species. To explain that difference, there are a lot of parameter to choose from during the construction of the network architecture. In the randomly initialized model, they were chosen a bit arbitrarily, because it was difficult, without previous experience, to know what parameter would be the best. Another factor is the size of the dataset. Indeed, a few hundred of picture per species is quite low to train a network and increasing the number of images the network can train on could also be a way to improve its accuracy. Also, the label given by the databases were considered as true, but it might not be the case. Particularly for Gbif, were the pictures are sometimes provided by amateur naturalist and the website provides no guarantee that the identification is correct.

Model	Global accuracy	Species	Accuracy
TL4.2	141/201 = 0.70	*Clarkii*	42/59 = 0.71
		Perideraion	23/25 = 0.92
		Ocellaris	19/23 = 0.82
		Frenatus	13/16 = 0.81
		Other	44/78 = 0.56

Table 5: Accuracy after training of TL4.2 on the validation dataset, globally and per species.

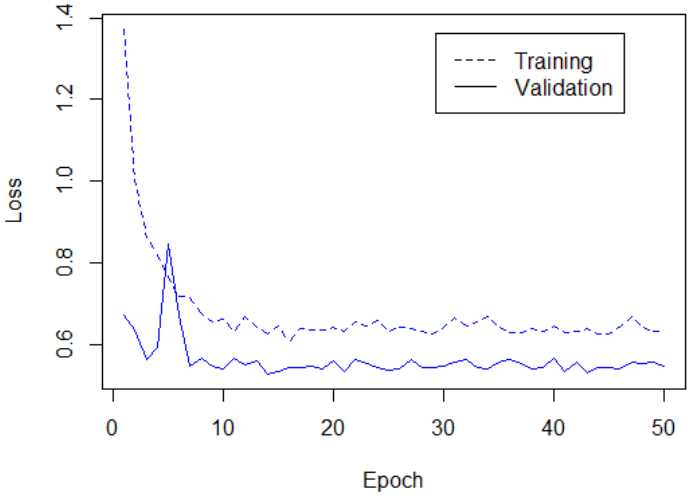

Figure 6:Change in loss during training for TL4.2

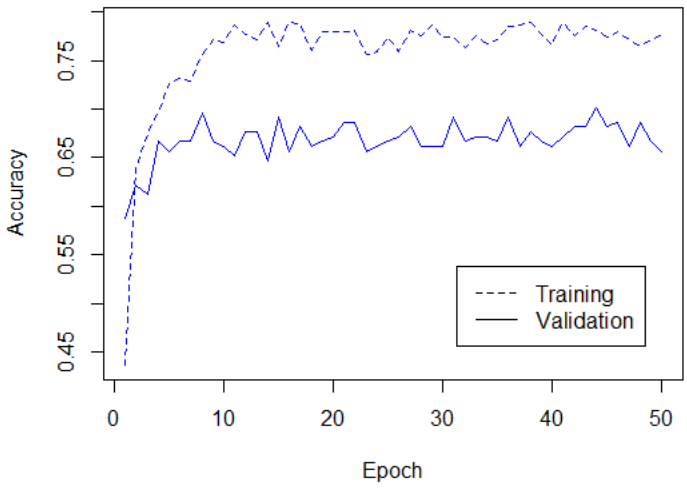

Figure 5: Change in accuracy during training for TL4.2

But to conclude on a positive note, the networks constructed in that first step project are working: training has improved their accuracy, they show no sign of overfitting and can predict the species of a clownfish on a picture way better than expected by chance and sometimes, like TL2 and TL4, with a quite satisfactory accuracy. These steps are the bases of the use of CNNs for image classification but are not by no means trivial.

Future perspective would be, to make these programs more applicable on large, not curated database would implement classification as "other" based on the score of confidence in prediction, as described in the result and discussion section for that network and if the quantity of data allows it, to increase the number of species classified.

IV. Method

a. Gathering images

The images were downloaded from three databases: Gbif, FishPix and Fishbase. The first allow to download a csv file containing the links to the images and the species name. Therefore the image were downloaded with a simple for loop and the urlretrieve() function of the urllib package. Pictures were saved in different folder according to the species name. However, Fishpix and Fishbase, are websites in HTML and accessing the direct link to each image was more difficult. On both websites, the provided search engine was used to query "Amphiprion". The python package beautifulsoup4.4.0, that allow to search and navigate more easily through the HTML code of webpages, was used to get the direct links to each image from the search result page. Then again, images were saved using urlretrieve(). The script used are available in the folder download images. Note that some images from Fishbase were manually removed of the dataset because they were drawing or even post stamps representing clownfish rather the real pictures. Some images from Gbif were also deleted, including pictures of clownfish bones, specimen conserved in ethanol and scans of observation report.

b. Prepare training and validation datasets

This subsection describes how:

- The images files were organized in a way usable by the torch package to train and validate CNNs
- The number of images in overrepresented Amphiprion spp was reduced

The Torchvision package contains several classes to create a python object that can be used as a database to train and validate a CNN. I chose to use the class Torchvision.dataset.ImageFolder(). It requires the image to be in a folder organized as follows:

filename/train/class1/
filename /train/class2/

...
filename /val/class1/
filename /val/class2/

To create such a folder, I first iterated through the whole image folder, counted the number of images per family and sotred it in a dictionnary. Two datasets were created, one with the two most represented species: A. clarkii and perideraion the other with four most represented species: A. clarkii, perideraion, ocellaris and frenatus. For each species, 10% of the images were randomly sampled for validation and copied to the validation folder (file/val/species). In an attempt to do a trade-off between a low number of images and an unbalanced number of images per families, the number of images per species in the training dataset was limited to two times more than the least represented species. It was done by randomly sampling a smaller number of image if the above condition wasn't fulfilled. Selected pictures were then copied to the training folder (file/train/species).

Before being input to the network, the picture were randomly sampled by batch of 16, loaded and transformed using torch.utils.DataLoader(). The transformations include: resizing image to match the size defined in the CNN architecture (96x96 for random initialization, 256x256 for transfer learning), converting the image from PIL image to pytorch Tensor and normalizing each color channel by the mean and variance of the whole dataset. For the training dataset, data augmentation steps were added. By applying some simple transformation to the input images at each epoch, the network tends to train more efficiently, as if there was more data. The first data augmentation step used here consists of flipping horizontally randomly selected images from the dataset. The second creates a crop of random size and a random aspect ratio (between 0.5 and 1 compared to the original). This crop is finally resized to the size expected as an input by the neural network.

c. Convolutional neural networks

This section will describe the different approaches used to program the CNNs. Variations include:

- Random initialization or Transfer learning.
- Number of family classified: two or four
- Version 1 or 2: with or without and additional class the labeled as "other" and containing randomly sampled pictures from all the other species.

Each network has a dedicated folder. The folder name summaries important characteristics of the model. For example, TL_2fam2 stands for Random initialization, two families, version 2. The script net*.py (replace the asterisk by the name of the model of interest) defines:

- The data transformation and augmentation

- The creation of the pytorch dataset object
- The data loader,
- The network architecture
- The training function

The prediction accuracy per species was calculated with stat*.py. The file trained*.pt stores the parameter of the trained network in a dictionary and can be loaded again to make new prediction or to further study the model accuracy. The bash script to run the program on the cluster is written in script*.sh. The change in loss and accuracy on the training and validation dataset at each epoch and the duration of the training process are stored in *.out.

All the networks were trained with the function train_model() adapted from the pytorch transfer learning tutorial. It takes as arguments:

- Model: an instance of a pytorch model. Has to inherit the class nn.Module
- Criterion: Defines how the loss function will be computed. Here, cross entropy was chosen. It is implemented in the function torch.nn.CrossEntropyLoss().
- Optimizer: Defines which algorithm will be used to calculate how to modify the network parameters to minimize the loss function (Backpropagation). Stochastic gradient descent [ref: Polyak] with learning rate = 0.001 and momentum = 0.9 was chosen. It is implemented in the function torch.optim.SGD()
- Scheduler. Applies modification to the optimizer at a given timepoint in the training process. Here, Pytorch will devide the learning rate by 10 (gamma = 0.1) every seven epoch (step_size = 7). Function : torch.optim.lr_scheduler.StepLR()

i. Random initialization

The architecture was based on the network by used by *Qin et al* [5]. However, the number of output class was adapted (two or four in this article vs. 23 in *Qin et al*, [5]). Also, the input image resolution was increased (96x96 in this article vs 46x46 in *Qin et al*. [5]) because in some of the pictures used the fish was small or hidden in the background. In addition, I was given access to the SCITAS cluster and therefore had the necessary computational power. The architecture of the network and the forward function are defined in the class Net96. The rest is inherited from torch.nn.module class.

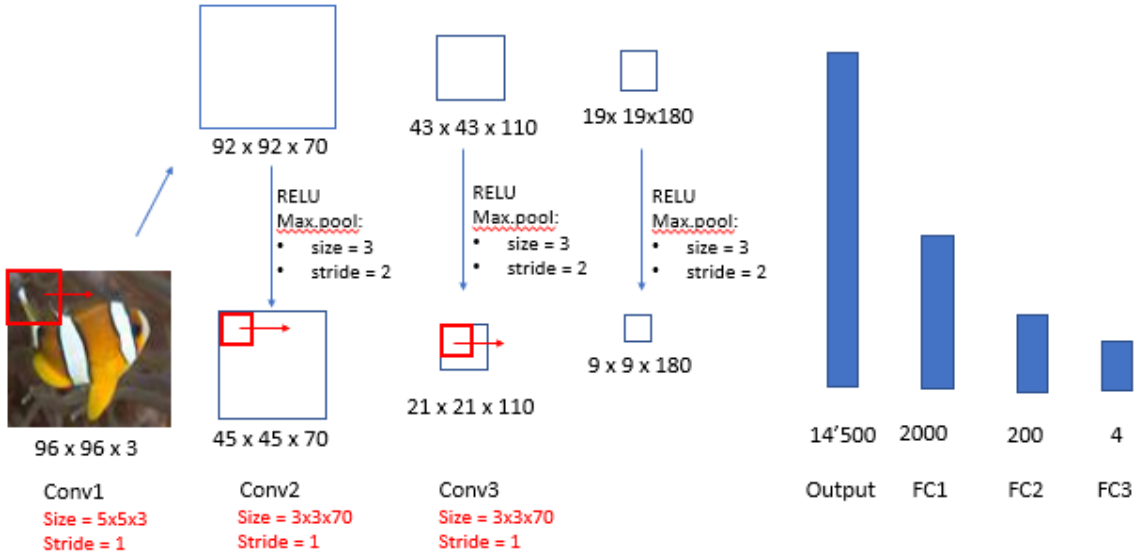

Figure 7: Schematic representation the randomly initialized network to classify four species. Images were passed through three convolutional layers. Filter size is 5x5 in the first layer and 3x3 in the second and third. Stride is set to 1. Each convolutional layer is followed by RELU and max pooling (filter size = 3, stride = 2). Convolutional layers are followed by three fully connected layers. FC1 and FC2 are followed by RELU. Output of FC3 is the number of species to classify the pictures in and is here set to 4. Architecture of the randomly initialized network to classify two species is the same, except the number of outputs of FC3, then set to 2.

The network was trained with the train_model() function described above, for 50 epochs. The training process was performed on the SCITAS cluster on 8 CPU and with 10G of memory. It took 20 minutes for two species and 30 minutes for four species..

ii. Transfer learning

The model used is this case is resnet18 [1] with filter and weights pre-trained on the Imagenet dataset [6]. This model was loaded using torchvision.models.resnet18() then all parameter were frozen, meaning further training won't affect them. An additional fully connected layer was then added with a number of outputs corresponding to the number of species to classify. The last layer is then trained on the dataset using train_model() for 30 epochs.. The training process was performed on the SCITAS cluster on 8 CPU and with 10G of memory. It took 22m 15s minutes for two species and 25 minutes for four species.

d. Results analysis

As said before, trained networks were tested in more in details on the validation datasets, with the scripts stat*.py. Also, the plots of the change in loss and accuracy were done using R. Data for these plots was extracted from the text files outputted by the training function.

e. Github

All the scripts are available at https://github.com/leojequier/Clownfish_recognition_CNN.

V. References

1.

He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]* (2015).

2.

Marcionetti, A., Rossier, V., Bertrand, J. A. M., Litsios, G. & Salamin, N. First draft genome of an iconic clownfish species (Amphiprion frenatus). *Molecular Ecology Resources* **18**, 1092–1101 (2018).

3.

Mariscal, R. N., Fautin, D. G. & Allen, G. R. Field Guide to Anemonefishes and Their Host Sea Anemones. *Copeia* **1993**, 899 (1993).

4.

Polyak, B. T. & Juditsky, A. B. Acceleration of Stochastic Approximation by Averaging. *SIAM J. Control Optim.* **30**, 838–855 (1992).

5.

Qin, H., Li, X., Liang, J., Peng, Y. & Zhang, C. DeepFish: Accurate underwater live fish recognition with a deep architecture. *Neurocomputing* **187**, 49–58 (2016).

6.

Russakovsky, O. *et al.* ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* **115**, 211–252 (2015).

VI. Softwares

R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Python Software Foundation. Python Language Reference, version 3.7.1. Available at http://www.python.org

VII. Packages.

A complete list of the package used with version is available in the file package.txt on github.

VIII. Databases :

Fishpix, Kanagawa Prefectural Museum of Natural History, Odawara & National Museum of Nature and Science, Tokyo. Link: http://fishpix.kahaku.go.jp/fishimage-e/index.html, consulted in october 2018.

Fishbase, Froese, R. and D. Pauly. Editors. 2018.FishBase. World Wide Web electronic publication. Link: www.fishbase.org, consulted in october 2018

Gbif.org, GBIF Occurrence Download, link: https://doi.org/10.15468/dl.z3nrp4, consulted on the 22th October 2018.