Future of the project

Table of Contents

Dataset

Acquisition

The first step, to get the networks to work in real condition, is the creation of a real database, with pictures taken on the mudflat. This could be done during missions that already exist, which would have the advantage of giving the possibility to labelise the data over time. Or we could organize a few missions especially to take these pictures.

The goal is to create a Database of approximately 1000 pictures taken on the mudflat, with different angles, different luminosity, different number of birds at different distances etc. To do so, it could be better to take pictures in the morning, in the evening and at dawn. An intermediary objective could be a database composed of images taken from the internet and on the mudflat.

Then, all pictures have to be labeled, as described in the Algorithms Doc. These Data could be augmented in some ways, and the Dataset for bird Detection would be finished. Note that it can be better to do the data augmentation during the training, in order to get more varied Data. However, it might make the training a bit longer.

To label the database faster, a script can be created in order to save the imperfect results of the neural networks. The required functions are in the ipynb present on github. Once the result is saved, LabelImage could be used in order to correct the data.

For the Object Detection Dataset, each bird present in the picture must be identified with a bounding box (tool provided in the algorithm doc). The bounding box must fit the bird perfectly, because the algorithm will use the similarity between the predicted bounding box and the bounding box given by the user (referred as ground truth).

For the Classification Dataset, the pictures would have to be cropped in order to have one bird on each picture. The goal is to reach at least 300 pictures for each species, and the Dataset could easily be filled with pictures found online (with an artificial diminution of the quality of the images). These pictures also have to be labeled as described in the Algorithms doc.

Training

The main thing that limited our training was Overfitting. Overfitting happens when the algorithm has trained too much on some particular data, gets very good results on it but then has horrible results on new data. We talk about the incapacity of the algorithm to generalize. To prevent this, we had to stop our training after a given number of epochs, but this limit of epochs can be improved thanks to a few methods :

- The more data you have, the longer you can train your model, but these data have to be varied.

- Adding Dropouts can be a solution to avoid overfitting, because it artificially and randomly adds “errors” to the network. You can check this link in bibliography **[1]** to know more about Dropouts. The problem with dropouts is it’s usually advised not to use them with Convolutional Neural Networks. We could see if adding Dropouts only to the Fully Connected part could be useful.

- Regularization is also a good solution to our problem. To sum up, the purpose here is to force the weights to be small, and it is proved to be a good way to prevent overfitting. Check about L1 and L2 regularization (Weight Decay) in bibliography **[2]** for more details.

All these things allow the model to search further for better solutions, without being blocked with an intermediate and insufficient one.

Algorithms

Classification

We used a pre-trained algorithm for the Classification, so one lead to improve our results could be to test other pre-trained models and compare them. Then, identify the structures that seem to be more efficient in our case and in the end code a model from scratch.

We also added a few layers to the pre-trained model, which is the part that we trained first. This part of the model is certainly improvable, by adding layers, either fully connected or convolutional, dropouts, new activation functions.

Changing the hyperparameters **[3]** (learning rate, optimizer, LR scheduler **[4][5]**…) can also be a good way to find improvement, but it will take a long time before having useful results.

A study of the impact of resolution on the results, in order to define the minimum amount of pixels per bird that we want on the images would also be interesting.

Detection

We managed to apply a neuronal network on images from ebirds. The neuronal network was trained on the COCO dataset. The results were encouraging, but this algorithm still performs badly in some situations. The first situation is when the birds are small, and the second situation is when the birds are close to each other. According to literature, this first problem should be solvable by using the right dataset. The second issue is harder to fix, and some situations will always be too difficult for reliable detection. However, an improvement is possible.

Even if we made it to apply an algorithm of detection, and trained it to detect birds in particular, we couldn't do much more than understanding the process, the different types of algorithms and technologies that can be used to make Bird Detection. There is still a lot of work to do on that part, and here are the possibilities for the next steps of the project :

- Create an algorithm from scratch, using existing methods but implementing them to create a new program. This could allow the algorithm to be more efficient in training time and results than others, because it would be designed for the particular task of detecting birds. It could also be the occasion to implement pre-processing methods such as using contours **[6],** or some other preprocessing methods **[7]**, in order to get better results. However, creating a complex algorithm like a faster-RCNN would be a huge amount of work. Furthermore, we don't have any assurance that it will work.

- Use an already existing algorithm, like we did, but have it trained on COCO fully transferred (using the API from tensorflow is too hard with ssd-resnet50). It would also take time to study it in detail, and adapt the process of training to get the best results. Still, it seems way easier than to create a program from scratch. There is some implementation of a neuronal network online in some gits using keras for tensorflow for exemple, which will be easier to use than the API of object detection.

<u>Here are some leads to get better results:</u>

Using masks could be a solution to the problem we have with birds that are close to each other. In each prediction bounding box, we would ask the algorithm what are the pixels corresponding to the bird. This information could be useful to the classification step, but also to count the birds (maybe thanks to this result, some fake predictions could be deleted because we will see the masks overlapping). Using this technique will require having masks for the dataset instead of bounding boxes. You can check the blog about Mask-RCNN for more informations **[9]**

Faster R_CNN should get the best results **[8]**. Faster R_CNN is the fastest algorithm with this level of results. We could get faster processing by using other algorithms such as YOLO, but the accuracy would be less good, and the time of processing is not an issue for us since we don't search to apply our algorithm on video datas. However, this algorithm takes a lot of resources (we didn't manage to train it on our 4Gb NVidia Geforce GTX 1650 due to a lack of available space). The solution to this problem is to train the algorithm in a computer with more GPU space or to use TPU, such as google colabs, where there is a free version. Distributed training can also be part of the solution, but i think it will be too hard to implement for a novice.

We think that using algorithms in Bird Detection is worth all these efforts, based on the results obtained by already existing Object Detection algorithms, and the one we used on flying birds. This is even more true when we consider it as a help for the user, who will be able, thanks to an IHM, to check the boxes created by the algorithm, erase some of those boxes or create new ones.

Utilisation

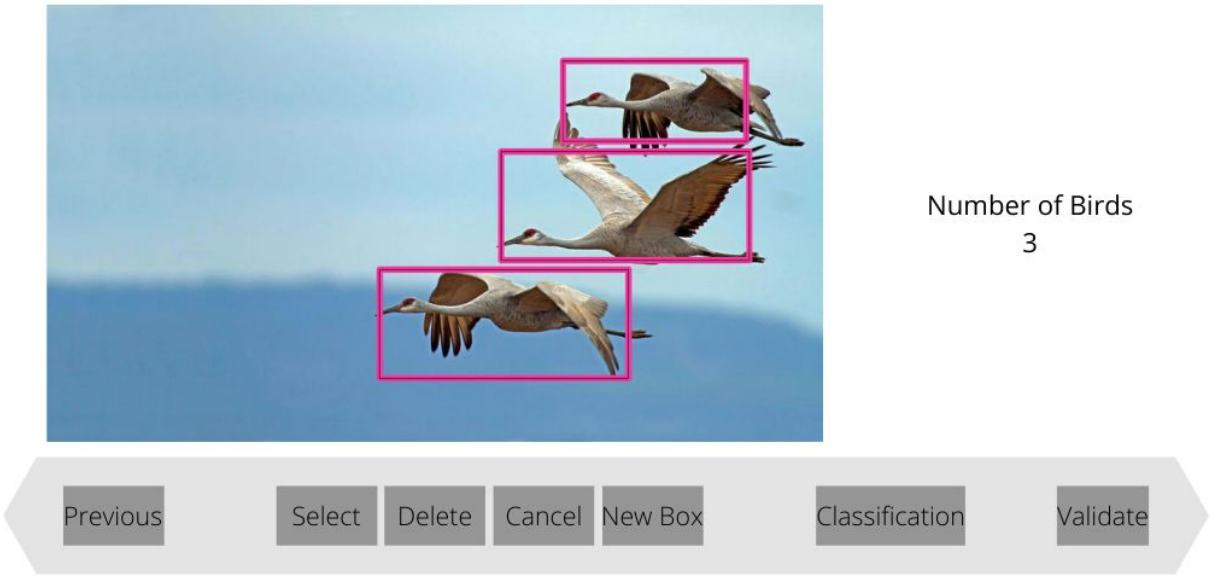
Introduction

The final product is made to help with the work on the field. Instead of counting the birds on the mudflat, one would only have to go and take pictures, and then analyze them later, with the help of the algorithms. Therefore, the user will have to make sure that birds are not on two different photos, if several photos need to be made in order to cover the entire area.

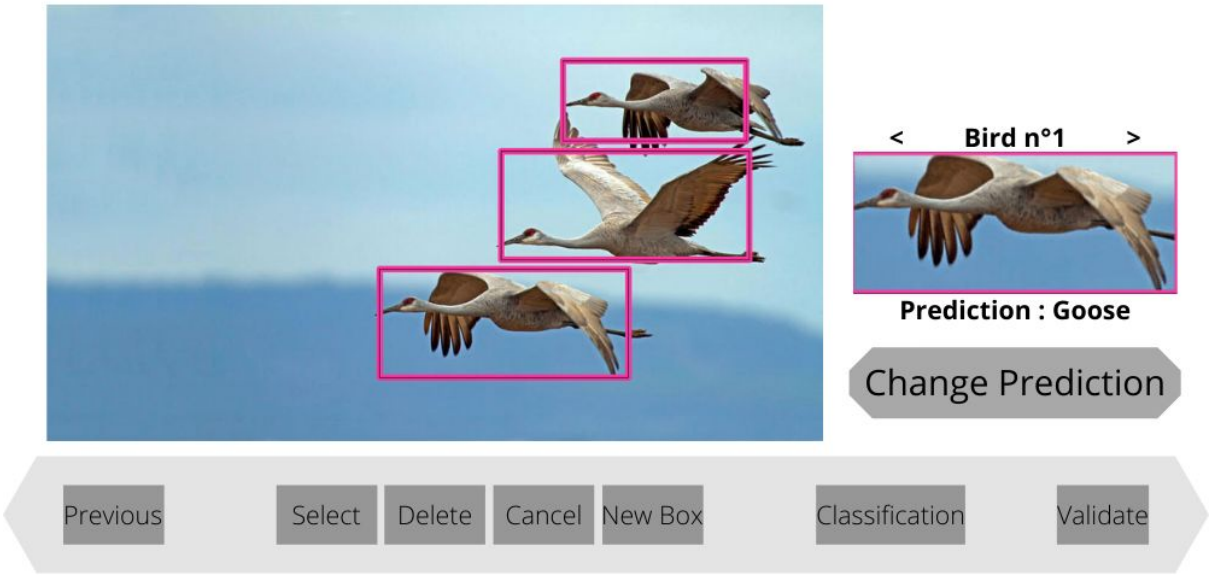
IHM

To make it easier to use, a simple IHM will have to be made. The application will ask for images in the first phase. Once the images are given, it is processed by the algorithm.

Then, the application plots the picture with all birds in boxes, and the number of birds for each species, to get a quick look at the result.



The user can either validate the result, or get a closer look on the classification by clicking on the corresponding button. The user will then be able to navigate among close up of birds, which correspond to the boxes created by the algorithm. These close up will be ordered by the confidence with which the algorithm gave a species to them.



The user can keep navigating among the close up, get back to the global picture or validate at any time. When it is validated, it is automatically added to the database.

Bibliography

N°	Subject	Links
1	Dropouts	https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
2	L1 and L2 Regularization	https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c
3	Hyperparameters	https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)
4	Learning Rate Scheduler explanation	https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1
5	LR Scheduler documentation	https://pytorch.org/docs/stable/optim.html
6	Contours	https://github.com/MichaelTeti/BirdDetectorCNN?fbclid=

		IwAR3aPfyPoXvAR_yGStucgHfJ4RGu1xNz14Uxsr9dzPjbfi-5pPB7ghC7SJo
7	Other Pre-processing methods	https://ijcsmc.com/docs/papers/May2014/V3I5201499a84.pdf
8	Detection methods	https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
9	Mask-RCNN	https://www.analyticsvidhya.com/blog/2019/07/computer-vision-implementing-mask-r-cnn-image-segmentation/?fbclid=IwAR2lkL864n_13RUrUUmMGERtDJqz1Npx6FP2waRwvxsCBs59L4lXRmqVXt0