20600 - Deep Learning for Computer Vision

Final Group Project

Group 5
Bagnara Arturo Mario 3188256 - Galfrè Michela 3076847
Marchetti Simone 3185524 - Marcianò Tancredi 3070110

December 6, 2022

Contents

1 Introduction

The goal of this project is to use transfer learning to perform an object detection task. We implemented two models: Faster R-CNN with a ResNet-50-FPN backbone and YOLOv7, using the first as a baseline. The aim is to detect and predict birds belonging to ten different species, comparing the performance of the two models, even in contexts in which the conditions of the images are significantly altered. In section 2, we describe the dataset, explaining the reasons for choosing the species to be analyzed. Moreover, the section examines the available data, providing visual representations. In section 3, we illustrate the two models used to solve this task, while in section 4, we show the various settings and alterations we made to the dataset, to trigger our models. Section 5 analyses the obtained predicted results. In section 6, we test the models on video and non-real images, to check models performance under odd conditions. Finally, section 7 reports the main key takeaways and possible future steps. Images and graphs can be found in the Appendix, while the source code and notebooks discussed in this report can be found at our GitHub repository `https://github.com/mgalfre/DLCV-project`.

2 The Dataset

Before showing the models we trained, we deem appropriate to firstly analyze the available data. We chose to use iNaturalist dataset[1], which provides labeled data of various types of plants and animals. There are in total 675,170 images and 544,900 boxes, relating to 13 animal species. We first discarded the species for which we had no boxes available. In the appendix (figure 1), a graph displaying the distributions of frequencies of the images available for each species can be found (among the ones for which we also have the boxes). We decided to work with Aves (birds), which is clearly the most represented class. The Aves super category features 964 subcategories. The goal of our project is to detect objects, recognizing the specific subcategory of the bird identified in each box. Working with 964 categories is prohibitively expensive, not only because of their amount, but also because some of these sub-categories have very few images, so training would be challenging. We therefore selected the ten most populated bird categories in the dataset: Melospiza melodia, Ardea alba, Pandion haliaetus, Cardinalis cardinalis, Zenaida macroura, Agelaius phoeniceus, Junco hyemalis, Ardea herodias, Buteo jamaicensis, and Picoides pubescens[2]. For each image we have the coordinates of the upper left corner, the width and height of the box and the label of the bird contained in the box, as shown in figure 2. From this bar plot it is also possible to see how unbalanced the classes are, if compared with each other. This could represent a potential problem in the predictive power of the models. To try to deal with this, as explained in section 4, we will try to implement undersampling and oversampling techniques. iNaturalist already proposes a division into train and val sets, which we have decided not to follow, for three main reasons:

1. iNaturalist subdivision is done into two sets (train and val), and this is incompatible with our requirements, because we need a split into train, dev, and test.

2. iNauralist's breakdown is excessively biased towards the train, with very few images being allocated in the val dataset.

3. the proposed val dataset is not exactly representative of the entire dataset, as only in the train there are images with more than one box, while in the val there are only images with one box.

Given these reasons, we therefore proceeded to perform a split train, val, test with 80%-10%-10% proportions, making sure that these proportions are preserved across the categories, using a stratified approach. Before training and testing the performance of the models, we perform a further data visualization step. We use CLIP model (Radford et al. [2021]) to obtain the embeddings of the images of the validation set. CLIP is a pre-trained model developed by OpenAI, which among other things allows to obtain representative 512-sized vectors of images. After vectorizing the images, we further reduced these vectors to 2 dimensions using PCA,

[1] See Git repository on `https://github.com/visipedia/inat_comp/tree/master/2017`

[2] The data selection steps that allowed us to select the ten sub-categories are available in the data_processing.ipynb file.

to be able to plot them in a scatter plot (figure 3) and to investigate the existence of patterns and similarities between images. From the plot, it is not directly straightforward to distinguish the various categories in the embedding space. The reason is that we are drastically reducing the dimensions (from 512 to 2), thus possibly missing relevant information. An interesting result that can be deduced, however, is that similar species are closely located in the embedding space, as it happens for Ardea alba (red) and Ardea herodias (pink).

3 Models

For object detection we need to build a model and teach it to learn to both recognize and localize objects in the image. To do so, we will perform transfer learning using Faster R-CNN and YOLO architectures.

3.1 Faster R-CNN

In our project, the first model we decided to use is Faster R-CNN (Ren et al. [2015]), a two-stage detector. The model we imported is a version pretrained on the COCO dataset using ResNet50. We will consider it as our baseline. Using this procedure to import the model we have the possibility to build the Dataset Class from scratch, according to our needs. Indeed, we defined the following two customized Dataset Classes and a Sampler Class:

- AvesAugmentationDataset: dataset class that satisfies our needs for the augmentations we will perform
- AvesStochAugDataset: dataset class needed for the under-sampled and over-sampled models
- SATMsampler: sampler class which defines the strategy to draw samples from the dataset homogeneously between classes

The reason why we need a personalized dataset class is because we want to perform data augmentations for our object detection task. Thus, it is not sufficient to use the standard torch transformations, but we need to use the Albumentations package (Buslaev et al. [2020]), which is a Python library for fast and flexible image augmentations. One of its key advantages is that it makes it possible to also transform boxes coherently with the image transformation. Some of the transformations we want to apply to our dataset are geometrical (e.g. HorizontalFlip, Rotate), while others work at pixels level (e.g. ColorJitter, Blur). We also defined a SATMsampler to pass as argument to the data loader, which defines the strategy to draw samples from the dataset, as anticipated, to aim to solve possible the problem of imbalanced classes. One more thing we have changed from the original model is the size of the image, bringing the max image size down to 242×242, balancing the tradeoff between efficiency and speed.

3.2 YOLOv7

The second model we use to perform transfer learning is YOLO (You Only Look Once) v7 model (Wang et al. [2022]), the latest version of YOLO single stage object detectors. To do so, we will use as reference the official GitHub repository by Kin-Yiu Wong[3], which we cloned and used for our object detection task. The main feature of this new version is definitely the execution speed, making it a state-of-the-art model. The procedure of having cloned the repository constrains us from being able to flexibly customise certain classes as we did with the Faster R-CNN. Despite this, we have seen that in the existing dataset.py files both Albumentations transformations and a DistributedSampler are used. However, in order to run YOLO we had to transform our annotations for labels and boxes to the format required. In fact, we have defined a dedicated function that transforms our annotations from COCO format to the YOLO one.

[3]See Git repository on `https://github.com/WongKinYiu/yolov7`

4 Settings

As anticipated, the Faster R-CNN model will constitute our baseline. We have decided to also include stochastic data augmentations to help the model retrieve useful features from the training dataset to better generalize to new data, and to make its results more comparable to the Kin-Yiu Wong's YOLOv7 architecture, where augmentations are already implemented automatically. Specifically, for each image we include two additional augmented versions on top of the original one, bringing sample size to 53415 (17805×3) images. As previously discussed, this was done through a custom dataset class named AvesAugmentationDataset where the number of augmentations and the desired transformations can be specified as input. The results of the Faster R-CNN and YOLO models will be discussed in the next section. Secondly, we will experiment performing undersampling and oversampling strategies to account for class imbalance (see figure 2).
As far as the Faster R-CNN model is concerned, our sampler class (SATMsampler) is given as an input to the dataloader object. It takes among its inputs the number of samples to draw from a source dataset (size), an array containing the labels for each image (labels) and the number of augmentations per image (num_aug). For each new sample, a label is drawn form a discrete uniform distribution ranging from 1 to 10, which will define its class. Secondly, an image belonging to the resulting class is sampled yet again from a uniform distribution over the space of all images belonging to that class. The process is repeated until a sample of the desired size is obtained. Notice that with this approach the samples will be automatically balanced between classes.
Over and undersampling is handled in the following way: the frequency of each class in the datasource is compared to the integer division between number of samples and the total number of labels (10 in our case). Each class that has a total frequency greater than this value will not be oversampled, as there already are sufficient images in the training dataset to produce a balanced sample. On the contrary, classes with a total frequency smaller than this value will instead have to be oversampled to balance the resampled dataset properly.
We will test 2 main settings: size = 10000 and size = 30000, keeping num_aug fixed to 3. In the first setting some classes will be undersampled as size = 1281× 10, while in the second setting all classes will be oversampled since size = 3627× 10, where 1281 and 3627 are respectively the frequency of least frequent class (Junco hyemalis) and most frequent class (Ardea herodias). Sampled images will be augmented through the custom class AvesStochAugDataset which takes as an input the argument prop_aug, the probability of applying random augmentations to each picture, making the results stochastic. Over and undersampling for YOLO are handled simply creating new directories with undersampled and oversampled images, where oversampled images were also augmented.

As a last inference task we decided to experiment with grayscaled pictures. Thus we trained both the Faster R-CNN and YOLO on grayscaled images and then evaluated their performance on the grayscaled validation set. This was done applying a grayscale transformation on all pictures for the Faster R-CNN, whereas new directories for grayscaled images were created to train and test YOLO.

As a final test we also examined the performance of our models previously trained on coloured images on the grayscaled validation set. Notice that models trained on three channels are not suited for images with a single channel, therefore two different approaches were tested:

- First we computed the average RGB value across channels for each pixel in every image, and cloned the same resulting one dimensional tensor (h,w,1) three times, once per RGB channel. Although this is not exactly equivalent to the grayscaled image from a technical point of view, the obtained image does not contain the information about the original contrast of colour, therefore represents an interesting test, as it allows us to explore how much our models rely on colours to detect and classify birds.

- For YOLO only we additionally experimented with a pretrained "DeOldify"[4] colorizing GAN, which tries to retrieve the original colours of the grayscaled images, so that we can feed the newly obtained pictures to our model. This allows us to explore to what extent this potential solution may affect inference, and possibly improve performance on new test colourless images.

[4]See Git repository on `https://github.com/jantic/DeOldify`

To conclude, we will test YOLO on abnormal new pictures that would never appear in real life with the aims to further unveil how the model makes its decisions.

5 Inference

In this section, we will compare two models and make inference on the results with the intent of getting some insight on what may be happening under the hood. The most used metric in object detection is mAP, which is considering the average precision across classes. However, to investigate more in depth each single class, we explicitly plot the confusion matrices and observe the recalls. Regarding the construction of confusion matrices, since YOLO provided them by default, we implemented them from scratch in order to be used also for studying FasterR-CNN. Notice that these are not standard confusion matrices, since we must account also for the predicted bounding boxes, meaning that we had to introduce some intermediate steps to check the Intersection Over Union (IOU) among the predicted and the true boxes.

5.1 YOLO vs Faster: base versions

Our baseline model is a Faster R-CNN trained with basic augmentations. Evaluating performance on our validation is: mAP YOLO = 0.83, mAP Faster = 0.763, IoU threshold = 0.5, objcteviness = 0.001.
In general the models are well behaved in terms of false negatives, which means a bird is usually not misclassified. Moreover, species are rarely missed, which results in low erroneous background prediction, suggesting that they're able to detect birds in the image.
However, we notice that Faster struggles slighlty more in distinguishing some pair of species: for example Pandion is often (10%) misclassified as Buteo (this is a reasonable result given that they are both predatory and someway similar birds), and Ardea Herodias with Ardea Alba (they have different colors, but very similar images). See figure 4.
YOLO has similar false negatives, still being more robust in classifying the correct class (all around 70%). Again, interestengly, we notice that Pandion is often incorrectly classified as Buteo. To further investigate this and we have a look at the classes distribution in the training, and indeed notice that Buteo appears roughly twice as many times (3328) with respect to Pandion (1794). This, on top of what we have previously discussed regarding the alikeness between these two classes, is suggesting we may want to account for this to improve the models performance. With respect to YOLO, it doesn't confuse the Ardeas species (see figure 5). Finally, in both models the best classified species is Cardinalis Cardinalis with a recall of 0.87 (Faster) and 0.88 (YOLO). An example of the predicted result of Faster and YOLO is displayed at figure 6.

5.2 Playing with the Sample Size

In this section, we address the class imbalances through two different approaches: *Undersampling* and *Oversampling*.

5.2.1 YOLO vs Faster: Undersampling

Both YOLO and Faster R-CNN display a significant drop in mAP with the previous strategy, respectively 0.73 and 0.668. This is expected, because we are reducing the global amount of training observations which means training procedure is harder since the number of parameters to adjust remained unchanged. However, since now all classes are uniformly distributed, we expect the under-represented species in the original dataset to have higher recalls. We can inspect the confusion matrices (see figures 7 and 8) and notice that in Faster, the rate of Pandion misclassified as Buteo indeed decreases. However, this doesn't seem like an optimal solution as now the model overly predict Pandion instead. Moreover, the overall performance drops as suggested by the mAP. Similar performance decay is observed in YOLO. This allows us to conclude although that rebalancing of the species distribution could contribute to reduce misclassifications for some classes, this is

not sufficiently counterbalanced by the heavy reduction in data observations. We therefore decide to test the oversampling strategy.

5.2.2 YOLO vs Faster: Oversampling

As far as mAP is concerned, comparing both models with their first configuration, oversampling leads to a slight decrease for Faster (0.746 vs 0.763), while there are no significant changes for YOLO (0.832 vs 0.831). Coherently with our expectations, the performance of YOLO on Buteo and Pandion has improved: oversampling the previously under-represented Pandion class allows the model to learn to classify this species more robustly, and the two classes are now more clearly distinguished, and the recall for Pandion rises from 0.68 to 072.
However, this is not the case for Faster: while it improves in distinguishing Ardea's species, it struggles while dealing with Pandion and Buteo. See results at figures 9 and 10.
The application of these strategies is suggesting us that, when the collection of new observations is not feasible, a simple oversampling approach may help the model to generalize to new data. Still, we must be cautious as the random oversampling may increase the risk of overfitting occurring, given that copying some observations may lead to the in-sample variance growing even more than the standard augmentation approach. On the other hand, although this is not the issue with undersampling, discarding observations and having less data to learn, the model risks to fail to find the correct decision boundary between the minority and the majority class.

5.3 Playing with the colors

Since both models behave like a form of black box, it is difficult for us to guess what key features are driving its decisions. However, we noticed that some species are characterized by strong colors (e.g. Cardinalis cardinalis which has bright red plumage), therefore we decided to "stress" the model training it on a copy of our images where we removed this information by greyscaling them.
A first option would be simply training both models on one channel and subsequently validate them on greyscaled data. Observing the mAPs we expect an overall drop in performance for the reasons we have just discussed (for example, the contrasts may become less distinct). Paying particular attention to the confusion matrices, and specifically focusing on species that originally display very vivid colors, it would be reasonable to expect the model to struggle detecting and classifying them.
Results are confirming our expectations: indeed for Faster mAP drops to 0.668 and YOLO to 0.74. However, this deterioration is not homogeneous among classes in both models, since we can observe that some classes (see figures 11 and 12) suffer more than others, for example Cardinalis and Junco, whose recall decreases by around 20% in both models. As far as the former one is concerned we suspect this may be due its color as we have previously discussed, while we must remember that the latter is the most underrepresented, which on top of the fact that we are also discarding part of the information may justify the negative impact on the performance.
Notice that with this approach we can infer that the models kernels are forced extract features regarding shapes and edges naturally independently from original unknown colors. Our results are suggesting there's still information to be exploited since recalls are still relatively high and significantly larger than 0.1, which is what we would expect by random guessing in a classical classification framework.

To analyze the matter of colors from a different perspective, we tested how our models trained on RGB images perform with the greyscaled set. For the technical details about the implementation see the end of section 4. We do so since our expectations suggest that a portion of the kernels might be devoted to extract features relative to colors, while another chunk may be connected to edges and shapes. This approach allows us to get a loose intuition about what fraction of kernels may address each of these two general purposes. Faster and YOLO mAP drop by 30 and 40 percent respectiveley, presenting an in insight into the importance of colors and suggesting that a large portion of kernels is now "useless", raising the suspect that our models may be overfitting the colors of the images. As a result (see figures 13 and 14) in fact Faster never correctly detects and classifies Cardinalis, while also YOLO struggles heavily for this species.

Finally, we experiment testing YOLO by recolorizing the testing images (for details see section 4). Unfortunately, the improvement is very minor (see figures 15). However this can be explained by the fact that the GAN used was not finetuned on our training dataset (for computational issues). Still, this strategy might become useful in the case where we have a model pretrained on RGB data that needs to be used on greyscale images, for example historical pictures.

5.4 Special cases

Following the insights of the inference so far, we will finally artificially create abnormal pictures we expect the model will find hard to classify. This experiments will be run exclusively with YOLO. Firstly we generated a picture where the head of a Cardinalis Cardinalis is morphed into the body an Ardea Alba (see figure 16). Despite there is one single generated bird, the model reacts to elements typical of both species and producing two seperate boxes, although significatnly overlapping: it detects the red head of the Cardinalis and the lower body of the Ardea. We suspect that the key feature bringing the model to detect Cardinalis is again its characteristic bright color. Therefore, to test this hypothesis we feed into the model the original picture of the Ardea, but changing its color to match that of Cardinalis (see figure 17). Indeed, again two partially overlapping boxes are detected, where the head and neck are correctly classified, while the main body as Cardinalis.
To conclude, we generated a new image featuring all of our species together (see figure 18). We are pleasantly surprised that YOLO is able to detect and correctly classify each class with relatively high objective score. Notice that there are no training instances with multiple different species.

6 Video Detection

Since YOLO is a state-of-the-art, real-time object detection system, we tried to run it over it a couple of videos just to see its performance over them. With respect to the Ardea's video, the model detects two classes: Ardea Alba and Ardea Herodias. We can consider this as a good result. Also, regarding the Cardinalis' video, results are better than expected, since both red and grey Cardinalis are detected. See videos with our predictions through the following links: `https://www.youtube.com/watch?v=0ILkHS5kYU8` (Ardea Alba) and `https://www.youtube.com/watch?v=Ks11U4TR5H4` (Cardinalis Cardinalis).

7 Final considerations

To wrap up our work, we run our base case models on a fresh new testing dataset, containing 2228 images. The mAP of YOLOv7 and FasterR-CNN are respecteviley 0.81 and 0.75 as shown in figure 19 and 20. These outputs are consistent with the result of prior literature. Notice that these values reflect those we had obtained on the validation set, proving that our splits of data didn't bias the results, making our inference generalizable. Moreover, YOLO outperforms Faster also in terms of testing speed, as it processes images roughly 3 times faster.

As far as potential ulterior improvements to our framework, we firstly suggest to try running both models on more epochs, as they are still not displaying heavy overfitting behaviors, and this could therefore further improve predictive performance.

Secondly, one could experiment with more sophisticated resampling techniques to account for the class unbalances, for example a Near Miss Sampling approach could discard pictures from the over-represented classes in a more efficient and legitimate way.

To conclude, to improve the performance over grayscaled images it could be advisable to fine-tune the "DeOldify" GAN model on our training data, which could be useful to reconstruct the colors of the original picture in a more sensible and informed way.

Bibliography

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. URL https://arxiv.org/pdf/1506.01497.pdf.

Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2):125, feb 2020. doi: 10.3390/info11020125. URL https://doi.org/10.3390%2Finfo11020125.

Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*, 2022. URL https://arxiv.org/abs/2207.02696.

A The Dataset

Get back to corresponding section: tap here 2.

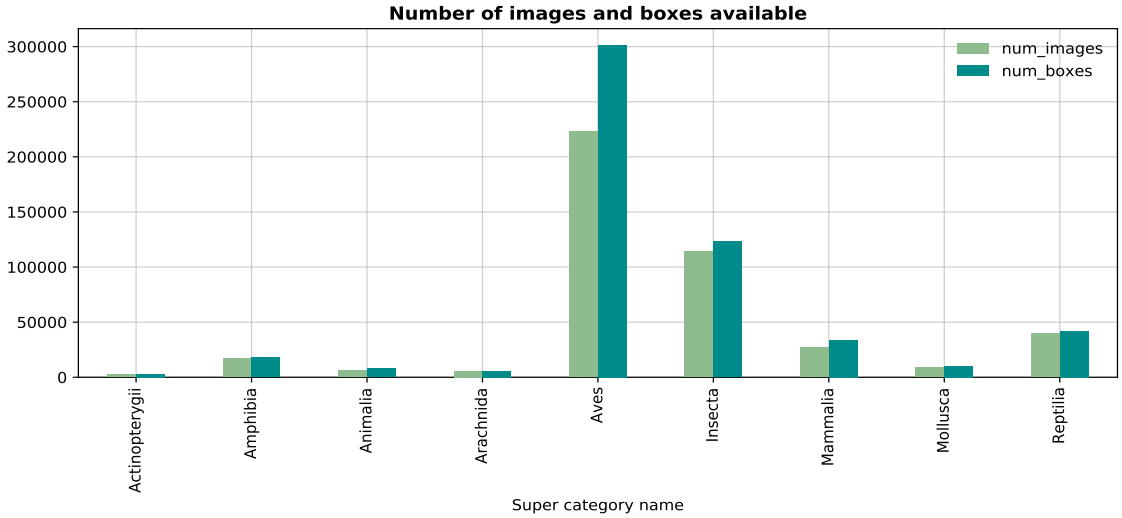

Figure 1: Images and boxes

Get back to corresponding section: tap here 2.

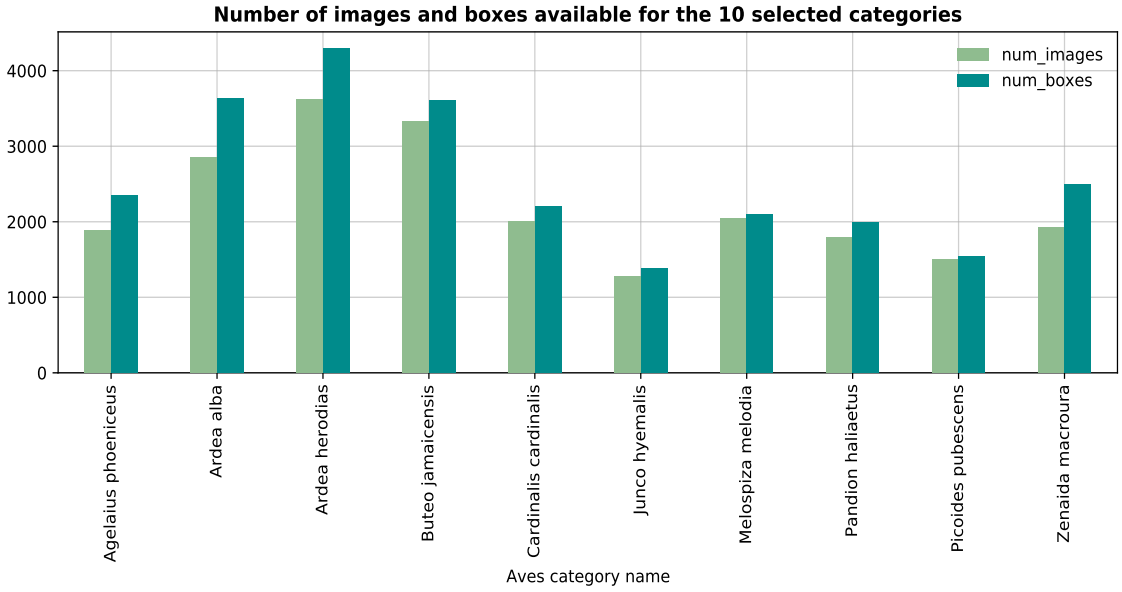

Figure 2: Selected Aves

Get back to corresponding section: tap here 2.

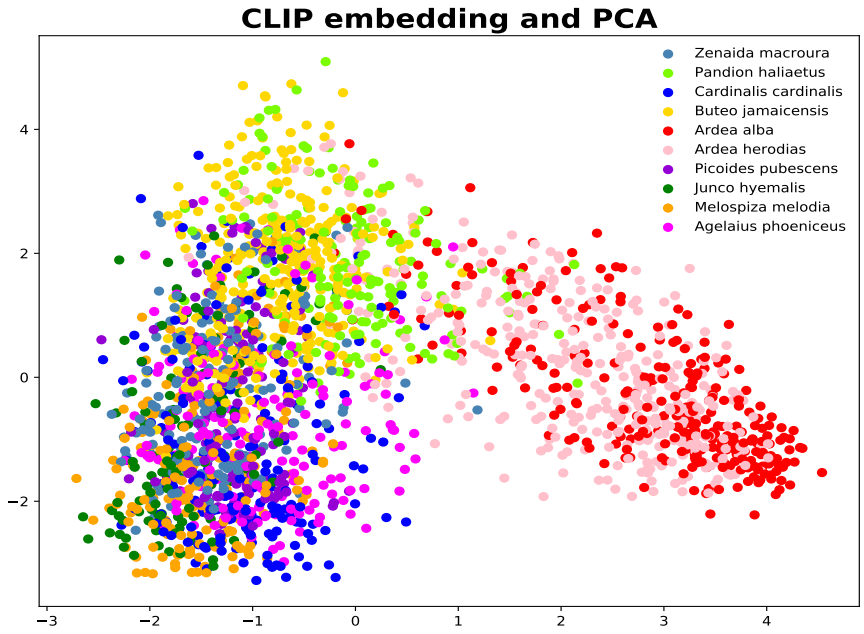

Figure 3: CLIP visualization

B Inference

B.1 YOLO vs Faster: base versions

Get back to corresponding section: tap here 5.1.

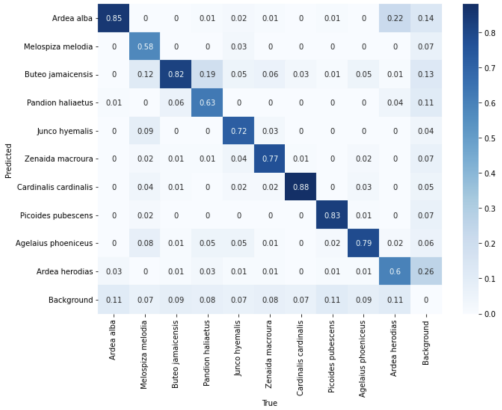

Figure 4: Faster R-CNN

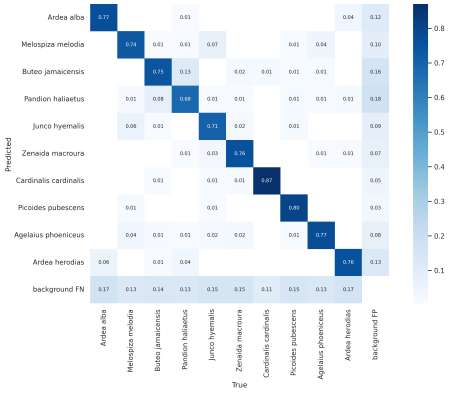

Figure 5: YOLOv7

Get back to corresponding section: tap here 5.1.

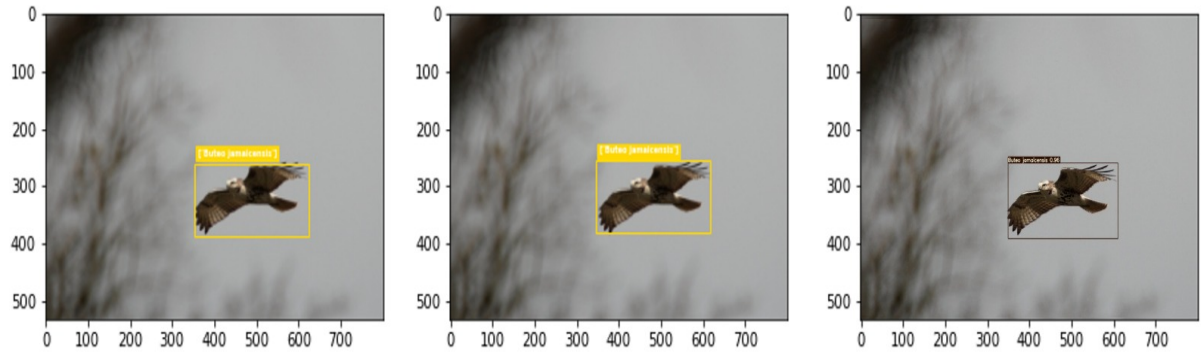

Figure 6: Ground truth, Faster, and YOLO predictions example

B.2 Playing with the Sample Size: Undersampling

Get back to corresponding section: tap here 5.2.1.

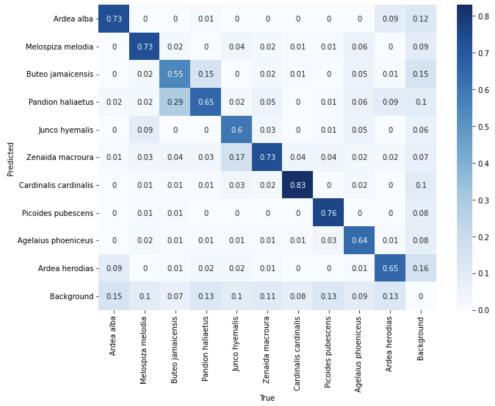

Figure 7: Faster R-CNN Undersampled

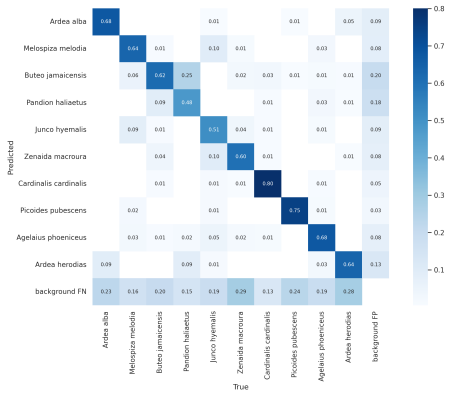

Figure 8: YOLOv7 Undersampled

B.3 Playing with the Sample Size: Oversampling

Get back to corresponding section: tap here 5.2.2.

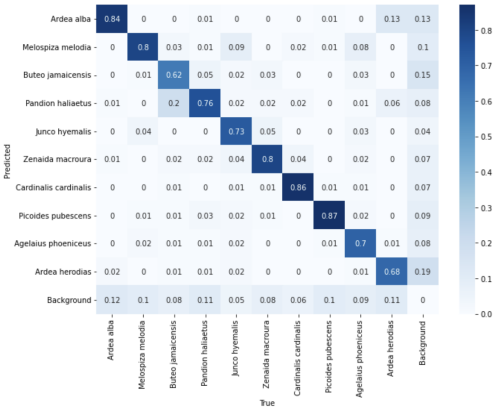

Figure 9: Faster R-CNN Oversampled

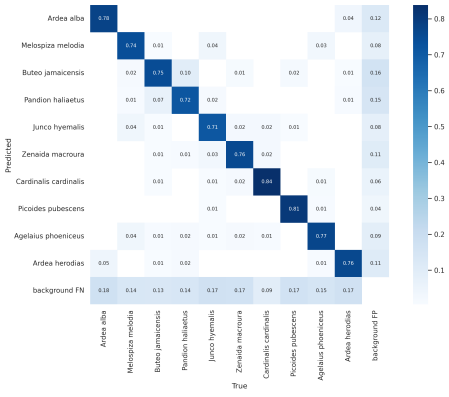

Figure 10: YOLOv7 Oversampled

B.4 Greyscale

Get back to corresponding section: tap here 5.3.

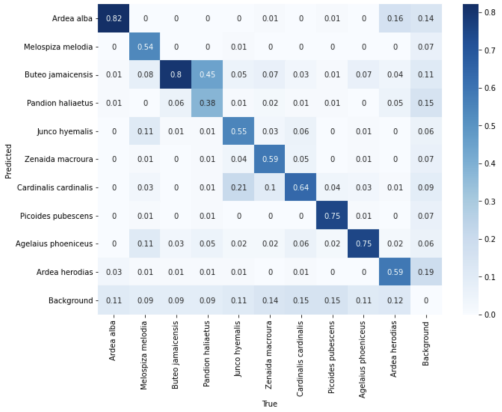

Figure 11: Faster R-CNN

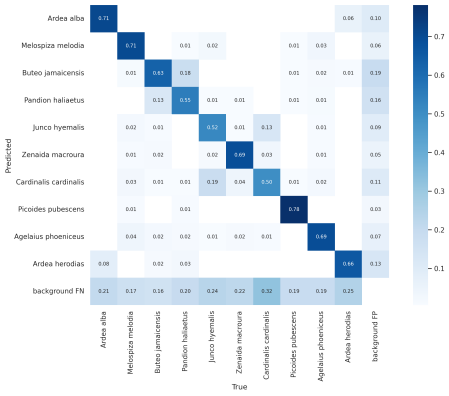

Figure 12: YOLOv7

B.5 Base models to Grayscale prediction

Get back to corresponding section: tap here 5.3.

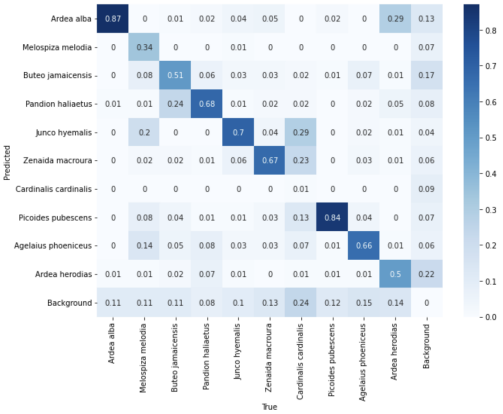

Figure 13: Faster R-CNN

Figure 14: Yolov7

B.6 Colorized Model

Get back to corresponding section: tap here 5.3.

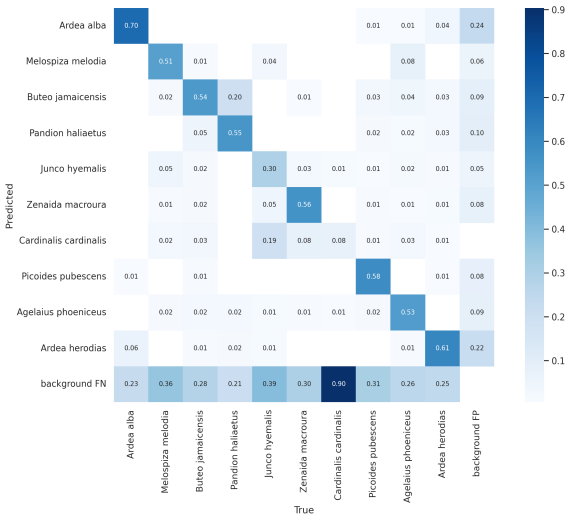

Figure 15: Yolov7

B.7 Special Cases

Get back to corresponding section: tap here 5.4.

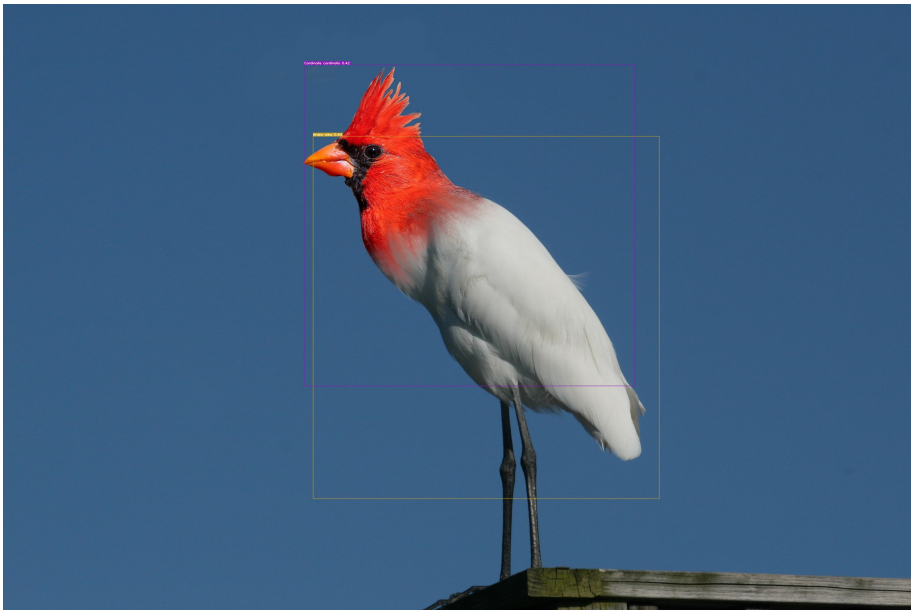

Figure 16: Paranormal bird version 1

Get back to corresponding section: tap here 5.4.

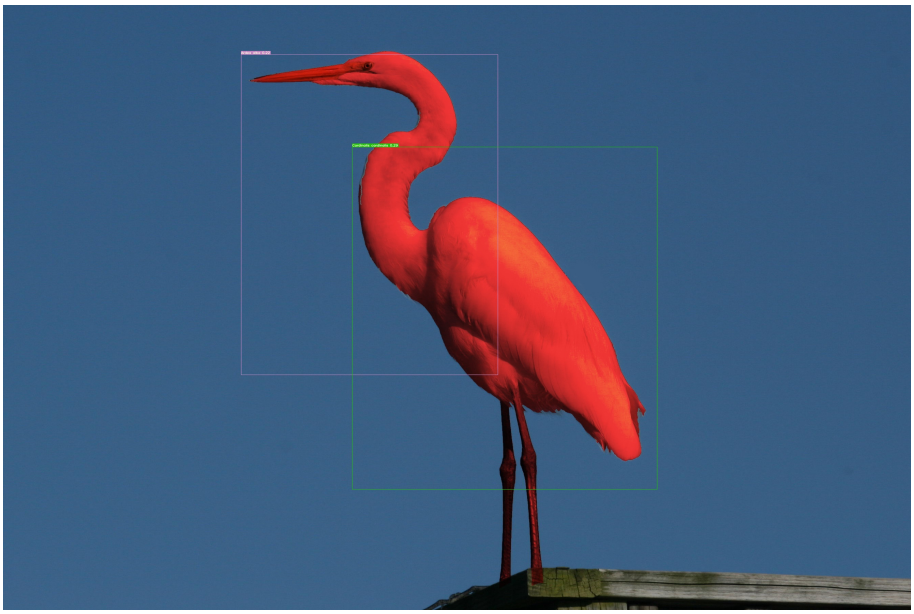

Figure 17: Paranormal bird version 2

Get back to corresponding section: tap here 5.4.

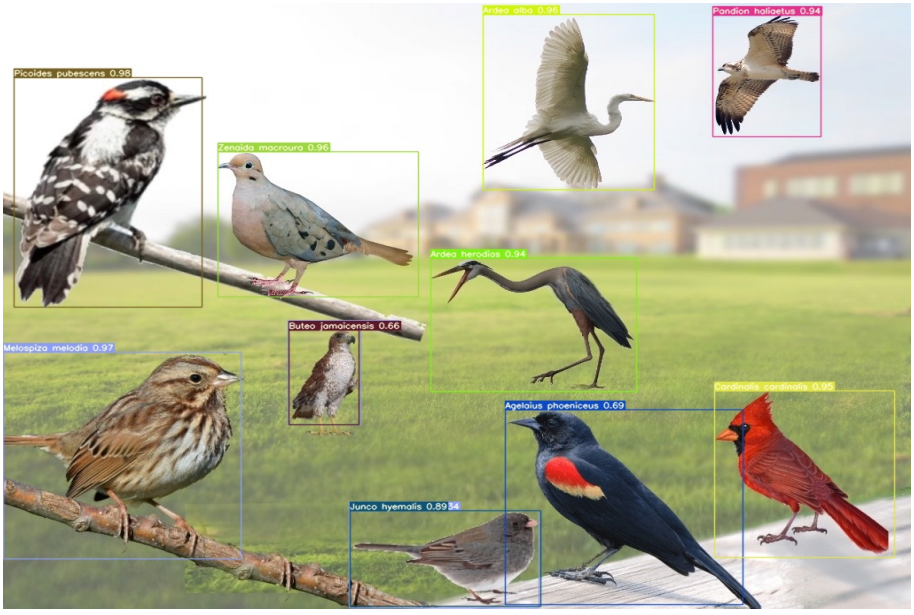

Figure 18: Flock

C Test

Get back to corresponding section: tap here 7.

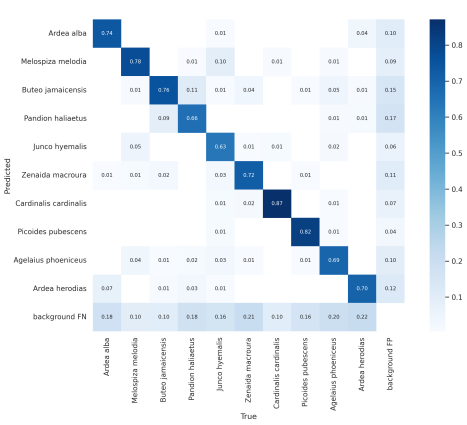

Figure 19: Confusion Matrix Yolo

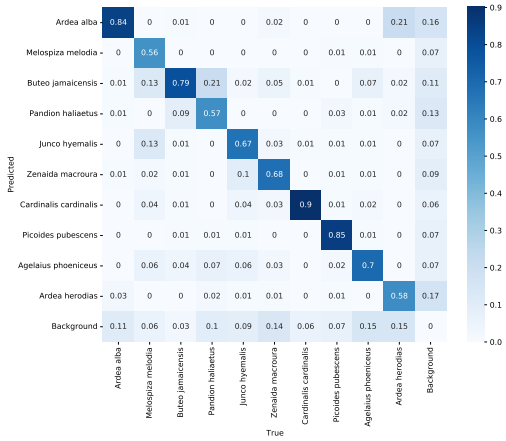

Figure 20: Confusion Matrix Faster