Few Shot Object Detection with CenterNet

Federico Cichetti
Department of Computer Science and Engineering
Alma Mater Studiorum Università di Bologna
federico.cichetti@studio.unibo.it
https://github.com/volpepe

Marcello Ceresini
Department of Computer Science and Engineering
Alma Mater Studiorum Università di Bologna
mceresini.97@gmail.com
https://github.com/MarcelloCeresini

Abstract

Few-Shot Object Detection (FSOD) is a complex problem to solve due to the limited availability of samples for the classes. One solution could be to decouple training into two phases: i) an initial large-scale pre-training on common objects that prepares the network for the complex task of object detection and ii) a subsequent fine-tuning on the objects of limited availability. PNPDet [1] achieves this by training a base detector and fine-tuning a "plug-and-play" head for few-shot classes. We re-implement this model and tackle the FSOD problem on the iNaturalist dataset [2], compiling an in-depth analysis of the data and implementing a novel algorithm for selection of novel samples. We thoroughly report our methodology and results. Code can be found at this GitHub repository.

1 Introduction and Related Work

In this work, we tackle the problem of Few-Shot Object Detection (FSOD), which involves detecting objects in images after training on a limited amount of data. For humans, this is a trivial task: looking at a new kind of object for a few times, or even a single time, is more than enough to remember its key features and be able to recognize it later. Instead, Object Detection algorithms usually rely on massive quantities of data in order to learn appropriate features for a class. Indeed, the detection accuracy of a specific class is typically positively correlated to the cardinality of images present in the dataset for the class.

Moreover, humans are consistently able to catalogue new objects while at the same time maintaining their knowledge about entities they frequently see in the world. For computers this might be a big problem, because incompatibility in layer sizes may require to start a new training from scratch each time a new object is introduced in the dataset. Also, the addition of too many objects may obfuscate previous knowledge, worsening performance on the frequent classes.

A comprehensive survey on FSOD [3] outlines the main challenges and possible solutions in the field, together with the common datasets and metrics used to evaluate the models. Formally, the problem can be defined in the following way:

> *The training dataset $\mathcal{D} = \mathcal{D}_{base} \cup \mathcal{D}_{novel}$ is split in two datasets of non-overlapping categories $\mathcal{C}_{base}$ and $\mathcal{C}_{novel}$.*

$\mathcal{D}_{base}$ is the dataset of common categories, for which we have a large amount of data, while $\mathcal{D}_{novel}$ contains all uncommon categories.
Each sample in the dataset consists in an annotation $A_i = \{I_i, c_i, b_i\}$, where I_i is the image containing the annotation, c_i is the category of the object, while b_i is the information on its bounding box, usually formatted as $\{x_i, y_i, w_i, h_i\}$ (x and y coordinates of its top-left corner and width and height of the box).
N-way K-shot detection is the setting in which $\mathcal{C}_{novel}$ contains only N novel categories, and each of the categories contains exactly K samples.

2 Dataset

According to [3], the two most used datasets for FSOD are PASCAL VOC (with 20 categories) and COCO (with 80 categories). For N-way K-shot object detection, datasets need to be split into $\mathcal{D}_{base}$ and $\mathcal{D}_{novel}$. In literature, to be able to compare the results of different approaches, some predefined subsets of categories are chosen: for example, in PASCAL VOC:

- Set 1: $\mathcal{C}_{novel} = \{$bird, bus, cow, motorbike, sofa$\}$
- Set 2: $\mathcal{C}_{novel} = \{$aeroplane, bottle, cow, horse, sofa$\}$
- Set 3: $\mathcal{C}_{novel} = \{$boat, cat, sheep, motorbike, sofa$\}$

In COCO, the 20 categories from PASCAL VOC are chosen as the novel set and the other 60 as the base set.

We think that this setting is not the best suited for FSOD, since the real average performance of the model is approximated only on some predefined subsets. Moreover, given our limited compute power, we wanted to start with a pre-trained ResNet backbone. Since backbones are frequently pre-trained on the entire COCO dataset, or even on datasets with many more samples and categories like ImageNet, the novel classes of both PASCAL VOC and COCO would not really be "novel" for the model.

Therefore, we started to consider another dataset: iNaturalist [2]. This dataset has two key advantages with respect to COCO and PASCAL VOC:

1. The distribution of the classes is more affine to this problem setting, as seen in Figure 1. Many classes in the dataset have more than some hundreds of samples, perfect to be eligible for the base section of the dataset. The novel dataset can instead be composed of those categories that only have a few tens of samples, which are almost half of the total categories.
2. Almost none of the classes in the dataset are present in ImageNet, allowing us to use the pre-trained version of ResNet to give a head-start to our research.

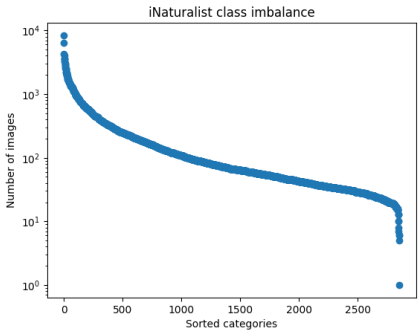


Figure 1: Distribution of samples per class in iNaturalist.

To be as general as possible with respect to the specific value of K used during training, we analyzed the data and chose two non overlapping groups of categories:

- $\mathcal{C}_{novel}$ is the group of classes having between 100 and 115 annotations in the dataset. We choose these thresholds in order to collect classes that are rare but at the same time have enough samples to build a substantial training, validation and test dataset.

- $\mathcal{C}_{base}$ instead contains the classes with at least 500 samples. The threshold was chosen as a trade-off between model performance and dataset usage, as it should allow the model to reach a good performance on base classes and at the same time avoid throwing large portions of the dataset away. With this threshold, we retain $51,37\%$ of the annotations and $48,01\%$ of the images.

In order to avoid any overlap between categories in our novel set and those present in ImageNet, we compared the names of the two lists of labels. We found only two exact matches: "*Panthera leo*" and "*Bonasa umbellus*". The two classes were removed from the novel set, but we acknowledge that other similarities among classes are possible. We feel that a more thorough exploration would be required to obtain more "realistic" novel subsets. We also highlight that ImageNet is known to have systematic problems related to annotations of wild animals. In [4], the authors found that many of the images representing wild animals in ImageNet-1k have not been closely scrutinised. This problem would make a thorough analysis even harder to achieve.

2.1 Dataset Exploration

Having separated $\mathcal{D}_{base}$ from $\mathcal{D}_{novel}$, we started to analyse the resulting datasets and subsets. In particular, we fix a training (B Train), validation (B Val) and test set (B Test) for the base dataset and check whether the distribution of data is similar between the different splits.

Table 1 shows some statistics for each split of the dataset. "Full", "Base" and "Novel" respectively refer to the original iNaturalist dataset, the sampled base dataset and the sampled novel dataset.

	Full	Base	Novel	B train	B val	B test
Categories used	2854	240	101	240	240	240
Images used	444319	213312	9328	170553	21216	21543
Annotations	544900	279913	10903	223837	27798	28278
Average ann/img	1.226	1.312	1.169	1.312	1.310	1.313
Min anns per cat	1	507	100	/	/	/
Max anns per cat	8386	8386	115	/	/	/
Images width	762	763	763	763	763	764
Width std	83	82	82	82	82	81
Images height	629	627	630	627	628	627
Height std	112	112	112	112	112	112
Ann/image area	9.79%	7.86%	11.77%	7.86%	7.82%	7.84%

Table 1: Dataset statistics for each split. On the rows, in order: number of categories, images and annotations in the splits, mean annotations per images, the number of annotations for the categories with the least and most annotations, images width and height (in pixels), their standard deviations and the average percentage of image area covered by annotations.

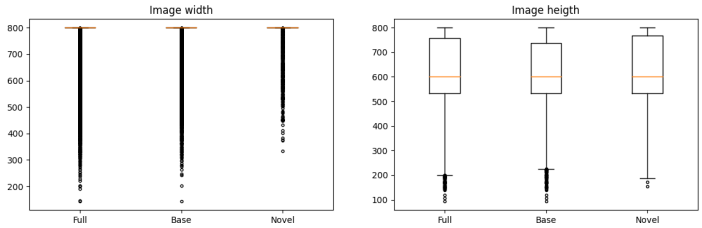
Figure 2b, 2a and 3 also show that all the different splits have similar properties. For example, the width, height of images and the bounding box areas are alike among all splits, which shows that images from the least represented classes have a similar structure to those from highly populated classes.

The only noticeable difference is in the ratio between annotation and image areas: in the novel set it stands at 11.77%, while in the base set it is only 7.86%. We hypothesise that either categories with less samples have more "close-up" shots of their subjects, occupying larger portions of the image, or that categories with more samples have a greater variance regarding the object sizes.

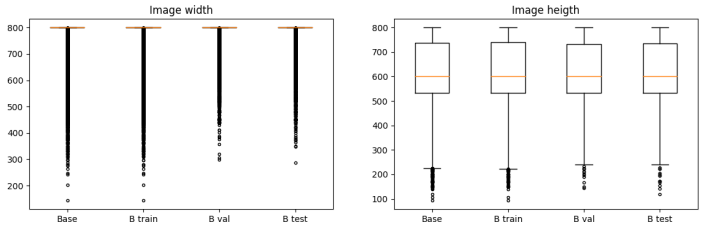
Finally, Figure 3 shows the distribution of cardinality of the categories in the different splits, with a focus on the novel dataset, and the gap between the novel and base splits.

2.2 Novel Dataset Creation

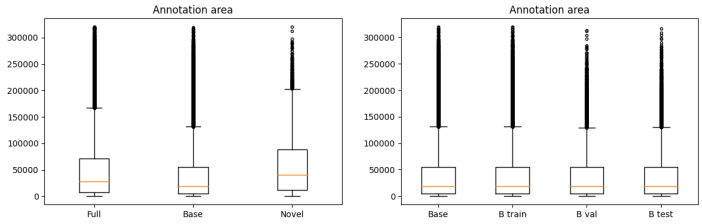
Following literature best practice, we decided to repeat the N-way K-shot training more than once and average the results. In fact, metrics may vary greatly depending on the sampled categories and on the specific K images that are used for training.



(a) Width and height of the images in the dataset



(b) Width and height of the images in the base splits



(c) Area of the annotations

Figure 2: Dataset statistics.

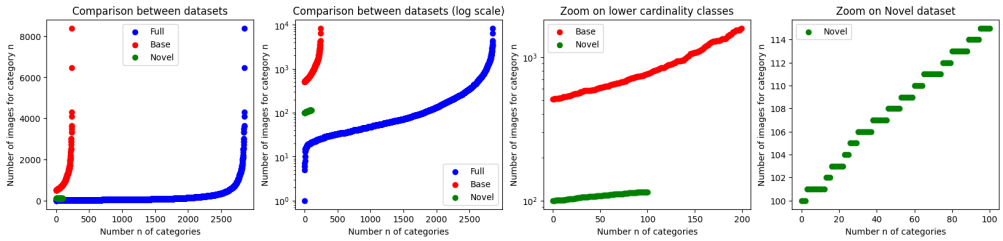


Figure 3: Number of images per category in the datasets

We repeat the selection of novel categories 5 times, in order to have a better overlook of the performance of the model on the whole distribution of novel categories. We fix $N = 20$ and explore different possibilities to look for a correlation between the performance of the model and the values of K (in $[1, 5, 10, 30]$). For this reason, we decide to implement a general approach and create the novel datasets online, without pre-defining the specific categories and samples used.

2.2.1 Algorithm for Online Novel Subset Selection

Our algorithm for online novel set creation takes as input N and K. It first randomly samples N novel classes from $\mathcal{C}_{novel}$ and assigns a random priority value to each of them. It also allocates a dictionary containing an empty set for each of the sampled classes. The idea is that of progressively filling up each set until they all contain K annotations.

We iterate over classes following the priority order, collecting for each of them all images that contain at least one annotation for that class. The collected images are also assigned a priority and, for each one in order, we take all of their annotations of classes in the set of sampled novel classes in consideration.

If adding all of these annotations to their respective class-sets makes one of the class exceed the limit of K sampled annotations, we skip the image and move to the next one. Otherwise, all annotations are added to the respective class-sets, and the algorithm proceeds with either the same class (if K annotations have not yet been collected for it), or to the next one.

This approach makes sure to avoid the risk of training the model on images that contain unannotated novel objects: in FSOD, all examples count!

At the end, we perform a final check: all classes must respect the constraint of K annotations or the dataset is re-built from scratch applying a different random priority for classes and images. A timeout is set to avoid that the procedure is repeated indefinitely. The process is repeated for the sampling the training, validation and testing novel sets, but each time we reduce the number of "sampleable" images for each class, avoiding to pick images that have already been used in other sets.

3 Model Architecture

Following [1] and [5], we implement a simple architecture composed by:

1. An encoder: inspired by [1], we choose a ResNet version (ResNet18 or ResNet50) pretrained on ImageNet. We cut off the classification heads that are not needed and keep all the layers until the last activation.
2. A neck: an up-sampling module composed by a series of deformable convolutions [6] and transposed convolution. It is needed to increase the resolution and reduce the number of channels, allowing for a more precise localisation while keeping computational costs low.
3. A regressor head: it outputs the values for the bounding box information (i.e. the size of the box and the offset with respect to the lower-resolution box center).
4. A classifier head for base categories: it outputs scores, pixel by pixel, for the base categories.
5. A classifier head for the novel categories: almost identical to the one above, but it is optional and outputs scores for the novel categories.

The 3 heads are implemented as a sequence of convolutional layers that refine the feature maps from the neck and output the correct amount of channels for our final heatmaps. Figure 4 shows the architecture of the whole model.

3.1 CosHead / AdaptiveCosHead

Distance Metric Learning (DML) is often employed in a few-shot classification setting. The goal is to learn representations of classes during training, so that it becomes possible to determine the category of testing images by comparing the features in terms of distance. However, object detection contains a background category which often overwhelms the samples of positive categories and is difficult to represent with a fixed set of features due to its variability.

To solve this problem, [1] introduces a module called *Cosine Similarity Comparison Head*. This module adapts DML to object detection by learning a representation space where features from the same category are closer than features from different categories. To shape this space, the model learns a *prototype* for each category and then compares them with features from different positions in the image using cosine similarity. In this way, the background category is implicitly learned as a point that is distant from all other prototypes and does not need a fixed representation.

The CosHead is a simple module that takes as input a feature map $F \in \mathbb{R}^{W \times H \times d}$ and outputs the final heatmap $C \in \mathbb{R}^{W \times H \times |\mathcal{C}|}$, where for each class, pixels have a value in $[0, 1]$ indicating the similarity between that location and the prototype of that class.

The results are computed as:

$$\mathcal{S}_{c,x,y} = \sigma(\tau \cdot \frac{W_c^T \cdot \mathcal{F}_{x,y}}{|W_c^T| \cdot |\mathcal{F}_{x,y}|})$$

Where both $\mathbb{F}_{x,y}, W_c \in \mathbb{R}^d$ and W_c is the learnable prototype for class c. Instead τ is a scalar set to 10.0, needed to extend the range and avoid too much compression from the sigmoid.

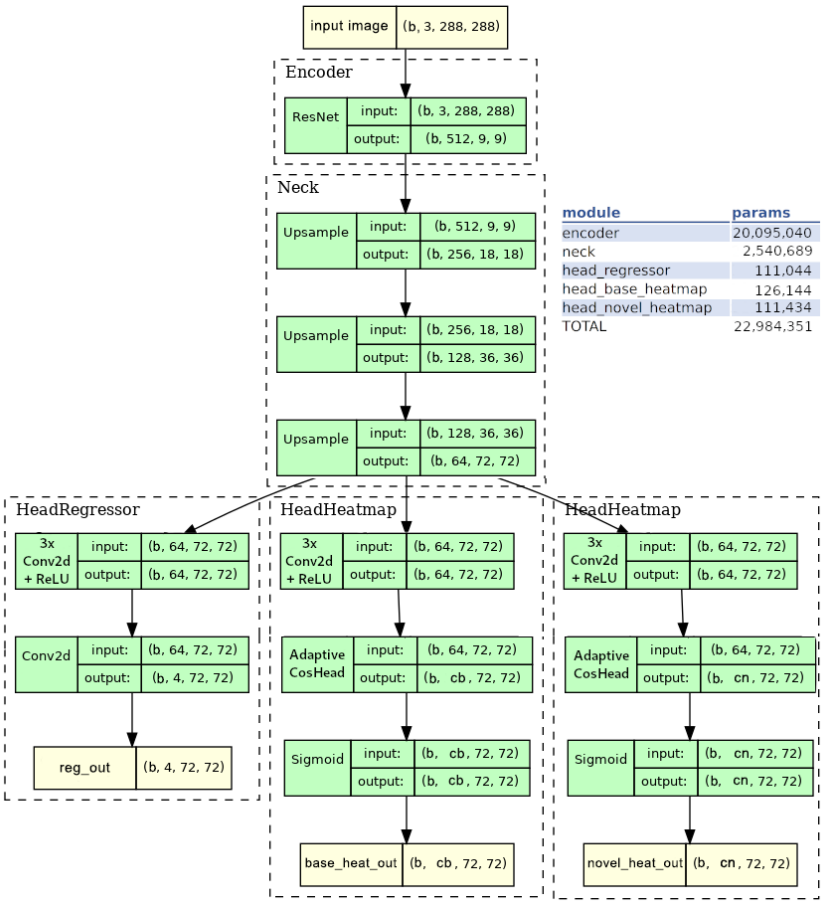


Figure 4: Model architecture. The `HeadHeatmap` branch producing an heatmap for novel classes is optional and removed during base training. `cb` and `cn` are the number of base and novel classes. The table on the right counts the number of parameters for each module.

This formulation, however, forces class with higher intra-category variance to suffer from a lower similarity value on average in their heatmaps, possibly leading to misclassifications, because of the functioning of our inference procedure explained in 3.6. To account for this, a simple variant named Adaptive CosHead is introduced:

$$\mathcal{S}_{c,x,y} = \sigma(\tau \cdot \tau_c \cdot \frac{W_c^T \cdot \mathcal{F}_{x,y}}{|W_c^T| \cdot |\mathcal{F}_{x,y}|})$$

Where τ_c is a learnable scalar for each class that enables learning of intra-class variance.

These modules are inserted at the end of the classification heads after the series of convolutional blocks, and are the ones responsible for the final activations.

3.2 Input Processing

During training, for both base and novel datasets, we augment the input with the following transformations:

1. Random Crop to the input dimension of the model. We make sure that at least one object is always maintained within the crop, retrying with different crops until the condition is satisfied.
2. Random Flip, both vertically and horizontally.

3. Color Jittering.
4. Gaussian Blur with a sigma randomly chosen in an interval and kernel size $= 7$.
5. Normalization into the range $[0, 1]$.
6. Per-channel normalization with mean $[0.485, 0.456, 0.406]$ and standard deviation $[0.229, 0.224, 0.225]$.

During inference we only resize the image to the standard input dimension of the model and follow the last two normalization steps.

3.3 Label Processing

Calling $I \in \mathbb{R}^{W \times H \times 3}$ and an input image to the model with annotation $A = \{I, c, b\}$, with bounding box information $\{x, y, w, h\}$, we create a low-resolution equivalent $\tilde{b} = \{\lfloor \frac{x}{R} \rfloor, \lfloor \frac{y}{R} \rfloor, w, h\} = \{\tilde{p_x}, \tilde{p_y}, w, h\} = \{p, s\}$ to produce three different labels:

- A regressor heatmap, $R \in \mathbb{R}^{\frac{W}{S} \times \frac{H}{S} \times 4}$, where $S = 4$ is the output stride, i.e. the ratio between input and output size, taken as default from literature. All the low resolution pixels are zero, except if they are in the center of a bounding box, no matter what category. In that case, they encode the low resolution $\tilde{b}$.
- A classification heatmap for the base categories, $B \in \mathbb{R}^{\frac{W}{S} \times \frac{H}{S} \times |\mathcal{C}_{base}|}$, that outputs values according to the presence or absence of objects of base categories.
- A classification heatmap for the novel categories, $N \in \mathbb{R}^{\frac{W}{S} \times \frac{H}{S} \times |\mathcal{C}_{novel}|}$, that behaves in the same way.

Both the classification heatmaps are constructed by creating bidimensional gaussians on the correct category channel around the low-res bounding box center of the object, using the Gaussian Kernel $Y_{xyc} = (\frac{(x-\tilde{p_x})^2 + (y-\tilde{p_y})^2}{2\sigma_p^2})$ where σ_p is a standard deviation that varies with the object size.

3.4 Losses

Calling $\hat{Y}_{xyc}$ the prediction of the model, we define the classification loss, for both heatmaps, as a pixel-wise (weighted during base training) logistic regression with focal loss:

$$L_{classification_w} = \frac{-1}{N} \sum_{xyc} \begin{cases} w_c(1 - \hat{Y}_{xyc})^\alpha \log(\hat{Y}_{xyc}) & \text{if } Y_{xyc} = 1 \\ (1 - w_c)(\hat{Y}_{xyc})^\alpha \log(1 - \hat{Y}_{xyc})(1 - Y_{xyc})^\beta & \text{otherwise} \end{cases} \tag{1}$$

$$L_{classification} = \frac{-1}{N} \sum_{xyc} \begin{cases} (1 - \hat{Y}_{xyc})^\alpha \log(\hat{Y}_{xyc}) & \text{if } Y_{xyc} = 1 \\ (\hat{Y}_{xyc})^\alpha \log(1 - \hat{Y}_{xyc})(1 - Y_{xyc})^\beta & \text{otherwise} \end{cases} \tag{2}$$

Instead, the regressor label is used for both the offset loss and the object size loss:

$$L_{offset} = \frac{1}{N} \sum_x y \begin{cases} |\hat{O}_{\tilde{p}} - (\frac{p}{R} - \tilde{p})| & \text{if } \tilde{p} \text{ is a keypoint} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$L_{size} = \frac{1}{N} \sum_x y \begin{cases} |\hat{S}_{\tilde{p}} - s| & \text{if } \tilde{p} \text{ is a keypoint} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

In both cases, an L1 loss is used and the loss accounts only for the keypoints $\tilde{p}$, all other pixels are ignored. Both the offset and the size are not normalized and/or rescaled, and we only use one map for all categories.

The final loss is a linear combination of the above:

$$L = L_{classification} + \lambda_{offset} L_{offset} + \lambda_{size} L_{size}$$

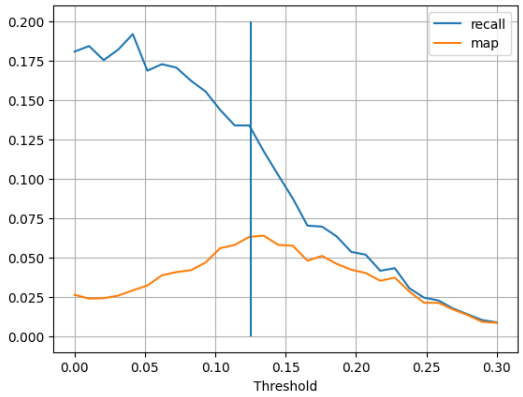


Figure 5: mAP and mAR with respect to classification threshold on the validation set.

3.5 Training Procedure

Training follows two main steps:

- Base training: we sample batches from $\mathcal{D}_{base}$ and disable the classifier head for novel categories. The loss is computed only on the regressor and the classifier heads for base categories. At the end of this training, the encoder, neck and heads reach their final weights that will be used during inference. During this part of the training, since the base classes are unbalanced ($16,54:1$ ratio between most and least represented categories), we add class weighing to the classification loss, where weights are the normalized inverse sample frequency of the classes.
- Novel training: following the procedure explained in 2.2.1, we create a small subset of $\mathcal{D}_{novel}$ with N random categories and K samples from each category that we then use to train on. In this phase we exclusively train the weights in the classifier head for novel categories, leaving all the others frozen. The weights in the convolution blocks of this head are initialized by copying the same weights in the other classification head. In this way, we warm start the training with parameters that already extract meaningful features from the image. The CosHead and Adaptive CosHead layers instead have to start from scratch and create their own feature prototypes.

The advantage of this training procedure is that the novel training is very fast, both because reduced number of learnable parameters and also for the size of the dataset.

3.6 Inference Procedure

During inference, the outputs of the two classification heads are stacked with each other, creating a complete heatmap containing both base and novel activations. The centers of the bounding boxes are chosen among the peaks of the heatmaps (found through 3×3 max pooling), and only the peaks with the top 100 scores are kept. Each of the 100 centers is assigned a central offset and a bounding box size thanks to the values found in the same (x, y) location on the class agnostic regressor head.

To improve metrics, we added a threshold to discard predictions if their confidence score is too low. This threshold is optimized on a small version of the validation set and is then used to evaluate the final model (see Fig. 5).

4 Results

4.1 Evaluation Metrics

The main metric we use for evaluating the efficacy of our model are Average Precision and Average Recall.

Given an Intersection over Union (IoU) threshold *T*, we order all the predicted bounding boxes by their score giving them a rank from 1 to *max_detections* and compare them with the ground truth boxes. If the IoU between a predicted box and ground truth box is higher than *T* and their labels match, it is considered a positive prediction, otherwise a negative one.

Moreover, if an object has already been detected, all the following positive detections are considered as negatives.

At each new predicted bounding box considered, we compute 2 values:

- Precision: the number of positive predictions over the rank of the bounding box
- Recall: the number of positive predictions over the total number of bounding boxes (in an image at first, and cumulatively over the entire dataset)

We then create the Precision-Recall curve and compute the Average Precision (AP) as the area under the graph:

$$AP@T = \int_0^1 p(r)dr$$

For the implementation we use *TorchMetrics* [7], a Python package that enables fast and reproducible metric calculations. In particular, we used *MeanAveragePrecision*, a function that accepts as input:

- Groundtruth boxes, with their spatial information and labels
- Predicted boxes, also with spatial information, labels and confidence scores

Then, it outputs the following metrics:

- map: mean average precision, where the mean is computed over the AP of all the categories and over multiple IoUs, in particular from .5 to .95 with 10 steps of .05.
- map_small: same as above, but for groundtruth boxes with area smaller than 32^2
- map_medium: same as above, but the area is $32^2 <$ area $< 96^2$
- map_large: area $> 96^2$
- map_50: the mean average precision only using $T = .5$
- map_75: a stricter metric, with $T = .75$
- mar_1: the final recall value when using only the top scoring prediction (averaged over the 10 IoU thresholds and over the all different categories)
- mar_10: same as above but using the top 10 predictions
- mar_100: uses the top 100 predictions, that in our case is exactly all the predictions
- mar_small: same as map_small but for recall values
- mar_medium: see above
- mar_large: see above

We also added two additional metrics to try and understand the nature of the predictions of the model:

- Localization metrics: we manually set all predictions and ground-truth bounding box categories to the same "0" category, allowing us to check the mAP of the model independently of the classification ability. We then feed all the results to the same function as above, giving us the same types of metrics.
- Classification metrics: since we cannot easily reduce the object detection problem to a classification problem, we decided to roughly estimate the classification ability of the model by considering only the boxes that had acceptable overlap: for each ground-truth box we compute the ratio between boxes that overlap with it (IoU $>= 0,5$) *and* correctly classify the object, and the total number of boxes that overlap with it. Then we average over all the ground-truth boxes in the dataset.

These two metrics help us understand if the model only tries to understand *where* the objects are or, when it actually understands the object position, if it also classifies it correctly.

4.2 Training Results

In this section we report our training results on the base and novel trainings. We utilise Weights and Biases [8] for logging and reporting on our experiments.

4.2.1 Base Training

We show an interactive report from Weights & Biases [8] that shows all the metrics of the three main base trainings we started. The main differences between the three are the following:

1. Resnet18, CosHead, Input Resolution $= [256, 256]$, Batch Size $= 4$
2. Resnet18, AdaptiveCosHead, Input Resolution $= [256, 256]$, Batch Size $= 4$
3. Resnet50, CosHead, Input Resolution $= [256, 256]$, Batch Size $= 4$

Base Training Report

Since the above results did not show significant improvements with the addition of the Adaptive-CosHead, we decided to continue the Novel Training part without utilizing it.

4.2.2 Novel Training

For the Novel Training, due to time constraints we chose to analyze only the first of the architectures above, with the following results:

Novel Training Report

After some trials, we understood that the same confidence threshold used during the base evaluation couldn't be used during novel training, since the activations of the model are much lower. We decided to reduce the maximum number of predictions of the model to 10, to match the mean number of predictions made by the base model in our evaluation on the test set.

5 Conclusions

In this work, we tackled a complex issue employing an innovative architecture and a unique methodology. Due to computational constraints, our results are unremarkable and we believe that there is much space for improvements. However, the training procedure resulted in a fairly satisfying behaviour: the quality of detections is notable, the ground truth class is often among the best predictions and the other predicted classes are mostly reasonable or fall within the same super-category.

Furthermore, we believe that our concerns regarding the fairness of the utilisation of datasets that have a large overlap with ImageNet-1k are valid, although iNaturalist is a very challenging dataset for FSOD.

Moreover, we structured a novel approach for a more sound evaluation of N-way K-shot learning, without the need of choosing predefined classes for the novel dataset and fixing the attention on the overall generalization capability of the model given K samples of any given class.

We intend to look into this matter further. Our future work can be structured around the following considerations:

- We could implement episodic training, to adapt our model to meta-learning (create subsets of the base database made just like the N-K-novel dataset that it will encounter in the second part of the training, and then train on them consecutively).
- Exploring other metrics, such as a top-k classification adapted to object detection or a super-class classification accuracy may help evaluating our model further in this hierarchical dataset.
- Adding more elements from metric learning (e.g. Triplet Loss) could help separate class prototypes of affine classes, reducing the misclassification errors.
- Adding normalization (with respect to input image resolution) to the size labels in order to improve generalization: in fact, with higher resolution input images the size of the bounding boxes would be greater than the range found in the training distribution.

- If the problem with overlapping boxes persists, we could add NMS to decrease drastically the number of final predictions and increase the mAP of the model.

References

[1] Gongjie Zhang, Kaiwen Cui, Rongliang Wu, Shijian Lu, and Yonghong Tian. Pnpdet: Efficient few-shot detection without forgetting via plug-and-play sub-networks. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 3822–3831, 2021.

[2] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The inaturalist species classification and detection dataset, 2018.

[3] Mona Köhler, Markus Eisenbach, and Horst-Michael Gross. Few-shot object detection: A survey. *CoRR*, abs/2112.11699, 2021.

[4] Alexandra Sasha Luccioni and David Rolnick. Bugs in the data: How imagenet misrepresents biodiversity, 2022.

[5] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points, 2019.

[6] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *CoRR*, abs/1703.06211, 2017.

[7] Nicki Skafte Detlefsen, Jiri Borovec, Justus Schock, Ananya Harsh Jha, Teddy Koker, Luca Di Liello, Daniel Stancl, Changsheng Quan, Maxim Grechkin, and William Falcon. Torchmetrics - measuring reproducibility in pytorch. *Journal of Open Source Software*, 7(70):4101, 2022.

[8] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.