# Few-shot Object Detection as a Service: Facilitating Training and Deployment for Domain Experts

Werner Bailer[1][0000−0003−2442−4900], Mihai Dogariu[2][0000−0002−8189−8566], Bogdan Ionescu[2][0000−0002−7728−0640], and Hannes Fassold[1][0000−0001−7113−0038]

[1] JOANNEUM RESEARCH – DIGITAL, Graz, Austria
{firstname.lastname}@joanneum.at
[2] National University of Science and Technology POLITEHNICA Bucharest, Romania
{firstname.lastname}@upb.ro

**Abstract.** We propose a service-based approach for training few-shot object detectors and running inference with these models. This eliminates the need to write code or execute scripts, thus enabling domain experts to train their own detectors. The training service implements an efficient ensemble learning method in order to obtain more robust models without parameter search. The entire pipeline is deployed as a single container and can be controlled from a web user interface.

**Keywords:** few-shot learning, object detection, ensemble learning, data preparation

## 1 Introduction

Few-shot object detection is useful in order to extend object detection capabilities in archiving and sourcing of multimedia content with specific object classes of interest for a particular organization or production context. For example, a specific class of objects may be relevant for a media production project, but not among those commonly annotated. Current workflows and tools for training few-shot object detectors have been created for data scientists and developers, but are not usable by domain experts (for example documentalists or journalists), who can provide and annotate the training samples of interest. Data scientists are usually not part of a media production team, and involving them is not possible on short notice for a small task. In order to leverage the progress in research on few-shot object detection for practitioners, it is thus necessary to enable them to use these tools themselves.

We build on the work in [2], which proposed integrated training scripts and support in content annotation tools. In addition, we provide ensemble learning methods for more robust training processes. Both are built using the same FSOD framework [14] and can thus be integrated. [2] has addressed the issue of integrating all necessary steps into the training workflow, and providing an extension of an annotation tool to label the data for novel classes. However, it may still be hard for non-experts to select an appropriate model and find a good configuration for training. In addition, once trained, the result is a serialised model, requiring access (via remote shell or desktop) to the training server in order to use the model further.

This paper addresses the shortcomings that prevent domain experts from training their own few-shot object detectors. It demonstrates a service for few-shot object detection that (i) runs the entire training workflow for the provided data, (ii) includes ensemble learning methods in order to facilitate achieving good results without the need for lengthy parameter tuning and (iii) deploys the trained model automatically as a service. The code is available at https://github.com/wbailer/few-shot-object-detection.

The rest of this paper is organised as follows. Section 2 provides a brief overview of the related work on ensemble learning for few-shot object detection (FSOD) and tools for few-shot training. Section 3 presents the integrated ensemble learning method, the training service is described in Section 4, and Section 5 concludes the paper.

## 2 Related Work

### 2.1 Ensemble Learning for FSOD

Few-shot object detection is by itself a relatively new topic, being addressed by a modest, albeit growing, number of works. As a sub-topic, ensemble learning in few-shot object detection is even more arcane. Ensembling efforts have been mostly made in the image classification field [7, 13, 1], or for object detection [5, 9, 4], in both cases approaching techniques such as cooperation, competition or voting schemes. Separately, different groups of researchers tackled the few-shot object detection problem [3, 11, 15, 14], with focus on meta-learning, weight sharing or fine-tuning approaches. However, we could not find any work related to ensemble learning in the few-shot object detection scenario. Therefore, to our knowledge, our work is the first in this field.

### 2.2 Tooling for Few-shot Training

Torchmeta [6] is a Python library for meta-learning, providing a common interface for working with different algorithms and datasets. However, no support for using own/aggregated datasets is provided. Argilla[3], a data framework for large language models, also provides some support for facilitating few-shot learning. Similarly, AutoML frameworks such as Amazon SageMaker[4] and AutoGluon[5] provide some support to set up few-shot learning tasks. However, all these approaches still require coding.

[3] https://argilla.io/

[4] https://aws.amazon.com/de/sagemaker/

[5] https://auto.gluon.ai/stable/index.html

The approach in [2] focuses on using custom data for novel classes, possibly in combination with other datasets. An integration with the MakeSense[6] annotation tool is provided, in order to obtain a set of files for the training process, which can be fed into a single script running the two-stage finetuning approach [14]. Two recent works for action recognition [8] and medical image segmentation [12] propose interactive few-shot learning frameworks, aiming at getting required annotations, as well as guiding the human annotator towards creating discriminative annotations. However, these frameworks do not consider deployment of the final model, and require having the human user in the loop.

## 3 Integrated Ensemble Learning

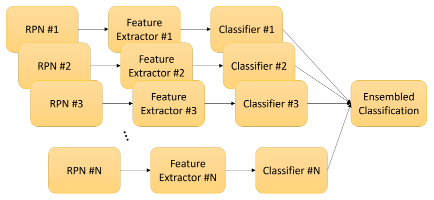
We base our ensembling strategy on a mixture between the works of Wang et al. [14] and Dvornik et al. [7]. In the former, FSOD is introduced as a fine-tuning step on top of a pre-trained two-stage object detector. The authors argue that having a pre-trained detector, it is sufficient to freeze the entire network, except for the last two layers and perform fine-tuning solely on these layers in order to obtain improved performance. In the latter, the authors tackle the problem of image classification and argue that having several networks performing the same classification task together yields better results due to having as little as possible different random weight initialization. The authors study the impact brought by having several almost identical classifiers perform the same job, with the only difference between them being the random values used to initialize the weights in the training process.

Two-stage object detectors generally consist of two fundamental sections: the region (or object) proposal network (RPN) and the feature extractor, together with a classifier, working on top of it. Ensembling strategies usually deploy several networks and process their set of outcomes, as depicted in Fig. 1a. However, this bears the cost of training N different networks with N usually being greater than 5. From the resource point of view, this type of processing is very costly, especially GPU-wise. Furthermore, the ensemble is usually distilled in order to reduce inference time, possibly at the cost of also reducing the system's performance.

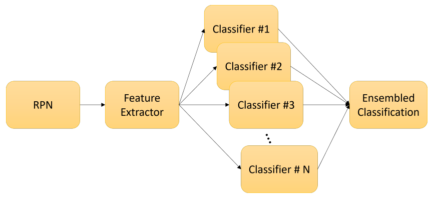
Our ensemble learning paradigm takes advantage of the fact that the framework presented in [14] freezes almost the entire two-stage object detection network. This leaves the object classifier part open for fine-tuning. Following [7], we apply a set of classifiers on top of the features extracted from the RPN's proposed boxes and generate N classification decisions for each proposed box, as depicted in Fig. 1b, thus approximately simulating an ensemble of N complete networks. Then, a regular non-maximum suppression (NMS) algorithm is applied for the resulting proposals. From this point on, the system approximates a single object detector, with enhanced detection capabilities.

The main difference between this method and the usual ensembling strategy is that we reduce the use of resources N-fold. One could argue that our pro-

[6] https://www.makesense.ai/



(a) Regular ensembling of FSOD systems.



(b) Proposed ensembling of FSOD systems.

Fig. 1: Comparison between regular and proposed ensembling architectures for FSOD systems.

posed system's RPN does not behave in the same manner as in the original case, having N times less proposed boxes to work on, but allowing the ensembled system's RPN to propose a large amount of possible objects (>2000) reaches the same performance as a combination of several RPNs, since the vast majority of the proposed regions are, in fact, not objects, and are therefore redundant. Another significant advantage of our method is that it is almost free to scale. Adding another network to the ensemble is reduced to adding another classification head to the architecture, which has insignificant impact from a memory standpoint. This method adds flexibility in the sense that it can be applied to a large number of network architectures that follow this working environment. Both the classifiers and the ensembling algorithm can remain unchanged from the regular ensembling setup. Performance-wise, our ensembling method adds a slight improvement to the detection performance of the original system [14]. To the best of our knowledge, this type of approach has not been tried before. Therefore, we compared our proposed system on the MS COCO [10] dataset with the original work of Wang et al. [14], while keeping the evaluation protocol unchanged. We obtained an AP@0.5 of 10.1 and 13.6 on 10 and 30 shots, respectively, compared to the original results of 10.0 and 13.4, respectively. Thus, our method essentially adds a marginal improvement with virtually no additional cost incurred.

## 4 Few-shot Learning as a Service

We aim to provide few-shot learning as a service usable by domain experts. The extension of the MakeSense annotation tool described in our earlier work [2] allows annotating the new images and downloading the configuration files for the few-shot training task. We extend this by providing a RESTful API for starting and monitoring the training process, together with a TorchServe[7] instance that provides a RESTful interface for running inference using both pretrained and new models. A simple web page for testing the whole process is provided, and

[7] https://pytorch.org/serve/

neither installation on the client side nor direct access to the remote server is required.

All components are deployed as a Docker[8] container to facilitate deployment. The container is built to be deployed on an environment with GPU acceleration. The scripts to generate and run the container are provided on the Github repository.

### 4.1 Training Workflow and Service

The training workflow builds on the script described in [2], which prepares the training and validation datasets and related configurations on the fly from the submitted data, runs the training on the novel classes, merges the base and novel class weights in the classification head of the model, and runs the finetuning over all classes. This script has now been extended to support the ensemble training methods, and integrated into the backend of a server exposing a RESTful interface, implement using Flask[9].

The training endpoint of the service receives two ZIP files, one containing the configuration and annotation files generated by the annotation tool and the other the set of images. The configuration file specifies the name of the model to be trained, base model used and the number of novel classes. These files are stored on the training server, and the training procedure is invoked.

In order to monitor the training process, a logging endpoint has been implemented, which can be invoked to access the entire log or a number of tail lines.

A simple HTML page is provided to invoke the training, view logs and test inference. The page is hosted by the Flask server providing the training endpoint. Figure 2 shows screenshots of different stages of this page.

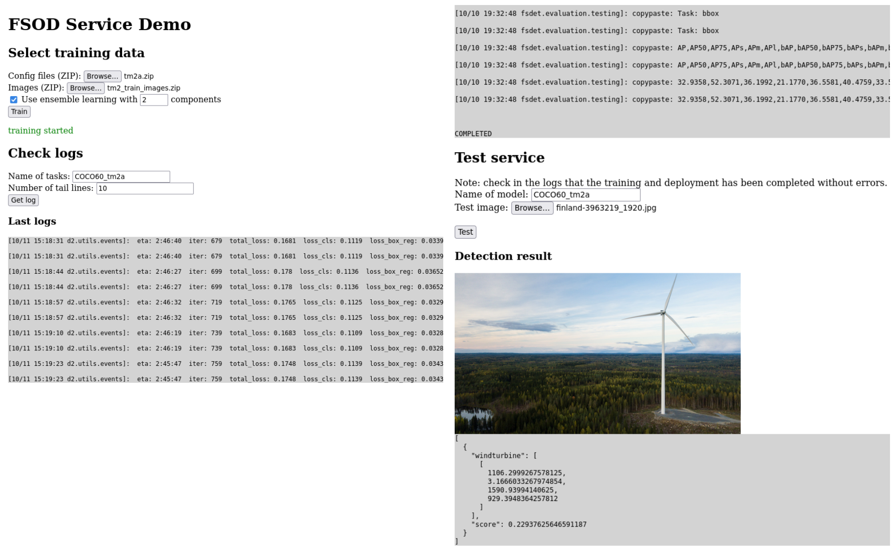
### 4.2 Auto-deployment

TorchServe is a framework for deploying PyTorch models for inference. A model is packaged as a model archive, containing a handler class, necessary configuration files and the trained weights of the model. We have built a custom handler based on TorchServe's object detection handler, which can generically serve the base models as well as any novel/combined models trained on the server.

Once few-shot training is completed, the required configuration files for the running inference in TorchServe are generated, and model archive is built. Using the TorchServe management API, the model is deployed to the server. To the best of our knowledge, this is the first work proposing a fully automated chain from training to deployment for few-shot object detection.

The TorchServe instance provided in the container starts already with base models trained on MS COCO, either on all or 60 of the classes (which is a common setting for evaluating few-shot object detection). Additional models are

[8] https://www.docker.com/
[9] https://flask.palletsprojects.com



**FSOD Service Demo**

**Select training data**

Config files (ZIP): Browse... tm2a.zip
Images (ZIP): Browse... tm2_train_images.zip
[✓] Use ensemble learning with 2 components
Train

training started

**Check logs**

Name of tasks: COCO60_tm2a
Number of tail lines: 10
Get log

**Last logs**

```
[10/11 15:18:31 d2.utils.events]:  eta: 2:46:40  iter: 679  total_loss: 0.1681  loss_cls: 0.1119  loss_box_reg: 0.0339
[10/11 15:18:31 d2.utils.events]:  eta: 2:46:40  iter: 679  total_loss: 0.1681  loss_cls: 0.1119  loss_box_reg: 0.0339
[10/11 15:18:44 d2.utils.events]:  eta: 2:46:27  iter: 699  total_loss: 0.178  loss_cls: 0.1136  loss_box_reg: 0.03652
[10/11 15:18:44 d2.utils.events]:  eta: 2:46:27  iter: 699  total_loss: 0.178  loss_cls: 0.1136  loss_box_reg: 0.03652
[10/11 15:18:57 d2.utils.events]:  eta: 2:46:32  iter: 719  total_loss: 0.1765  loss_cls: 0.1125  loss_box_reg: 0.0329
[10/11 15:18:57 d2.utils.events]:  eta: 2:46:32  iter: 719  total_loss: 0.1765  loss_cls: 0.1125  loss_box_reg: 0.0329
[10/11 15:19:10 d2.utils.events]:  eta: 2:46:19  iter: 739  total_loss: 0.1683  loss_cls: 0.1109  loss_box_reg: 0.0328
[10/11 15:19:10 d2.utils.events]:  eta: 2:46:19  iter: 739  total_loss: 0.1683  loss_cls: 0.1109  loss_box_reg: 0.0328
[10/11 15:19:23 d2.utils.events]:  eta: 2:45:47  iter: 759  total_loss: 0.1748  loss_cls: 0.1139  loss_box_reg: 0.0343
[10/11 15:19:23 d2.utils.events]:  eta: 2:45:47  iter: 759  total_loss: 0.1748  loss_cls: 0.1139  loss_box_reg: 0.0343
```

```
[10/10 19:32:48 fsdet.evaluation.testing]: copypaste: Task: bbox
[10/10 19:32:48 fsdet.evaluation.testing]: copypaste: Task: bbox
[10/10 19:32:48 fsdet.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,APl,bAP,bAP50,bAP75,bAPs,bAPm,
[10/10 19:32:48 fsdet.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,APl,bAP,bAP50,bAP75,bAPs,bAPm,
[10/10 19:32:48 fsdet.evaluation.testing]: copypaste: 32.9358,52.3071,36.1992,21.1770,36.5581,40.4759,33.
[10/10 19:32:48 fsdet.evaluation.testing]: copypaste: 32.9358,52.3071,36.1992,21.1770,36.5581,40.4759,33.


COMPLETED
```

**Test service**

Note: check in the logs that the training and deployment has been completed without errors.
Name of model: COCO60_tm2a
Test image: Browse... finland-3963219_1920.jpg

Test

**Detection result**

```
[
  {
    "windturbine": [
      [
        1106.2999267578125,
        3.1666033267974854,
        1590.93994140625,
        929.3948364257812
      ]
    ],
    "score": 0.22937625646591187
  }
]
```

Fig. 2: Screenshot of the demo HTML page when configuring training and viewing logs (left) and testing inference with the newly trained model (right).

deployed and reachable via an endpoint that is named with the label provided for the model in the configuration file.

## 5 Conclusion

Building on previous work for facilitating training of few-shot object detectors, we provide the entire training and testing workflow of detectors as a service, usable for domain experts without the need to write code or install any software on the client device. The training service also integrates an efficient ensemble learning method.

**Acknowledgements** The research leading to these results has been funded partially by the European Union's Horizon 2020 research and innovation programme under grant agreement n° 951911 AI4Media (https://ai4media.eu).

**Disclosure of Interests** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Afrasiyabi, A., Larochelle, H., Lalonde, J.F., Gagné, C.: Matching feature sets for few-shot image classification. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9014–9024 (2022)

2. Bailer, W.: Making few-shot object detection simpler and less frustrating. In: International Conference on Multimedia Modeling. pp. 445–451. Springer (2022)
3. Bar, A., Wang, X., Kantorov, V., Reed, C.J., Herzig, R., Chechik, G., Rohrbach, A., Darrell, T., Globerson, A.: Detreg: Unsupervised pretraining with region priors for object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 14605–14615 (2022)
4. Carranza-García, M., Lara-Benítez, P., García-Gutiérrez, J., Riquelme, J.C.: Enhancing object detection for autonomous driving by optimizing anchor generation and addressing class imbalance. Neurocomputing **449**, 229–244 (2021)
5. Casado-García, Á., Heras, J.: Ensemble methods for object detection. In: ECAI 2020, pp. 2688–2695. IOS Press (2020)
6. Deleu, T., Würfl, T., Samiei, M., Cohen, J.P., Bengio, Y.: Torchmeta: A Meta-Learning library for PyTorch (2019), https://arxiv.org/abs/1909.06576, available at: https://github.com/tristandeleu/pytorch-meta
7. Dvornik, N., Schmid, C., Mairal, J.: Diversity with cooperation: Ensemble methods for few-shot classification. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 3723–3731 (2019)
8. Gassen, M., Metzler, F., Prescher, E., Prasad, V., Scherf, L., Kaiser, F., et al.: I3: Interactive iterative improvement for few-shot action segmentation. In: 2023 32nd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), Busan, SOUTH KOREA (2023)
9. Lee, J., Lee, S.K., Yang, S.I.: An ensemble method of cnn models for object detection. In: 2018 International Conference on Information and Communication Technology Convergence (ICTC). pp. 898–901. IEEE (2018)
10. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13. pp. 740–755. Springer (2014)
11. Liu, F., Zhang, X., Peng, Z., Guo, Z., Wan, F., Ji, X., Ye, Q.: Integrally migrating pre-trained transformer encoder-decoders for visual object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 6825–6834 (2023)
12. Miyata, S., Chang, C.M., Igarashi, T.: Trafne: A training framework for non-expert annotators with auto validation and expert feedback. In: International Conference on Human-Computer Interaction. pp. 475–494. Springer (2022)
13. Tian, Y., Wang, Y., Krishnan, D., Tenenbaum, J.B., Isola, P.: Rethinking few-shot image classification: a good embedding is all you need? In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16. pp. 266–282. Springer (2020)
14. Wang, X., Huang, T., Gonzalez, J., Darrell, T., Yu, F.: Frustratingly simple few-shot object detection. In: International Conference on Machine Learning. pp. 9919–9928. PMLR (2020)
15. Xiao, Y., Lepetit, V., Marlet, R.: Few-shot object detection and viewpoint estimation for objects in the wild. IEEE Transactions on Pattern Analysis and Machine Intelligence **45**(3), 3090–3106 (2022)