Few-Shot Object Detection in Valorant using Multiplicative Layer-wise Learning Rates

Idris Wardere, Du Huang, Adrita Das, Alice Gatera

Abstract

Currently, object detection models require a lot of image data to achieve great results. However, in settings where labeled data is limited, traditional models may struggle to perform well. This is especially true in online video games where developers constantly add and alter content, leading to a constantly evolving environment. In this project, we seek to apply few-shot object detection methods in Valorant, a first-person game set in a three-dimensional environment developed and published by Riot Games in order to address this challenge. We develop a detector that can quickly adapt to novel categories with limited training data while leveraging the knowledge gained from a large base set with abundant annotated instances. To accomplish this, we fine-tune a pre-trained Faster-RCNN object detector with 20 images from 10 different Valorant characters using three different methods: training with no frozen layers, training with all frozen layers except for the last, and our proposed method, training with multiplicative layer-wise learning rates (MLLR)[1]. We evaluate the performance of each method using a custom dataset consisting of 20 images from 10 different characters. We find that our method performs the best out of the three methods on our dataset. This project has the potential to enable more efficient and accurate object detection in the gaming industry, where rapid adaptation to changing content is essential

Index Terms: Few-shot object detection, computer vision, deep learning

1. Introduction

Modern object detectors can achieve satisfying performance with the help of an enormous amount of data, but what if we only have a few samples? This project aims to learn a detector that can collect transferable knowledge from a large base set with abundant annotated instances (aka base categories) and adapt quickly to a novel set with only a few instances (aka novel categories). Note that the base categories and the novel categories are mutually exclusive. Then after training, the detector is required to do well in novel object detection. The goal of this project is to apply few-shot object detection methods in Valorant, a first-person game set in a three-dimensional environment developed and published by Riot Games. We build and pre-process our own dataset by taking screenshots from within the game. To tackle the few-shot object detection task, some researchers decided to take a pre-trained object detector and freeze all of its layers except the final one for the purpose of fine-tuning and achieving state-of-the-art results publication.[2] The intuition behind this strategy is that we only need to modify the final layer because the earlier ones are sufficiently trained for object detection. We can interpret freezing layers as setting their learning rates to zero. The previously described method can then be imagined as having one layer with some non-zero learning rate and all previous layers being defined as zero. What if we instead made the learning rates gradually decrease starting from the final layer and going towards the earlier layers rather than only defining a non-zero learning rate for the final one? This is the idea that we begin to explore using our proposed fine-tuning approach, multiplicative layer-wise learning rates (MLLR) [1]. Code is available on `https://github.com/idriswardere/fsod-valorant`.

2. Related Work

Few-shot object detection is a field that has been recently getting a lot of attention. In the past few years, there have been many very different approaches to how to improve this area. In addition, the object detection field is very closely related to it. Understanding object detection as a whole is critical to building a strong understanding of this task.

2.1. Object detection

The most closely related is regular object detection. This task is relatively simple to understand: given an image, produce bounding boxes and classifications for each object in the image. The images within object detection datasets are typically labeled with such information. There have been many approaches to this task, including using a Transformer-based architecture [3], fully convolutional networks with multi-level predictions [4], anchor-box-based approaches [5], and architectures with assistance from region proposal networks [6]. Each of these approaches adds its unique way of tackling the object detection problem.

2.2. Few-shot object detection

It has been shown that typical object detection approaches usually don't perform well in the few-shot setting right away, so many different approaches have been formulated to try to make models that can perform well in few-shot settings. One such work [7] was able to achieve state-of-the-art results upon publication by using an Attention-RPN and a Multi-Relation detector along with contrastive training. However, some [2] have shown that just fine-tuning the final layer of existing object detection can produce state-of-the-art results. More recently, one line of work [8] produced state-of-the-art results using a Transformer-based architecture [3] with an unsupervised training strategy focused on pre-training on object localization with a region proposal network and object embedding. Many approaches using meta-learning were also attempted [4].

2.3. Faster R-CNN

The Faster R-CNN is an object detection system that consists of two modules(as shown in Figure: 1) - a fully convolutional network that proposes regions and a Fast R-CNN detector that utilizes the suggested regions. It is designed to function as a unified network for object detection [6]. In the first step, the image is fed through a convolutional neural network (CNN) backbone to obtain a feature map. The ground truth bounding boxes are then projected to the feature map, and each point of the map serves as an anchor point for generating different sizes and shapes of anchor boxes. A 1*1 convolutional network is used to predict the category and offsets of these anchor boxes, and during training, positive and negative anchor boxes are selected based on the overlap with the ground truth boxes. In the second stage, region proposals are generated from the positive anchor boxes and passed through another CNN to predict the category of the object. These region proposals are resized using ROI pooling and aligned with the ground truth boxes using a network that predicts offsets. Both category and prediction are learned simultaneously using multi-task learning. Finally, a weighted combination of the losses from both stages is used to compute the overall loss.

The R-CNN primarily functions as a classifier and does not have the ability to directly predict object boundaries, except for minor adjustments through bounding box regression. Its precision and accuracy are therefore heavily influenced by the effectiveness of the region proposal module. Faster R-CNN and RPN (Region Proposal Network) were foundational technologies used in many of the top-performing entries in the ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation tracks of the ILSVRC and COCO 2015 competitions[9] [10]. In other words, these two technologies played a crucial role in the success of many winning entries in these computer vision competitions.

2.4. Loss Functions for Object-detection using Faster R-CNN

Loss functions in object detection are important since it affects the detection precision in these type of tasks. The loss functions for object detection can be classified as classification loss and localization loss. The classification loss is mainly used for training the classify head for determining the type of the target object. It is based on the Softmax function and measures the difference between the predicted class probabilities and ground truth class probabilities. It is calculated for each RoI (region of interest) proposal in the image. The goal is to correctly classify each RoI as either containing an object or not.

On the other hand, localization loss is used for training another head for regressing a rectangular box to locate the target object. The localization loss measures the difference between the predicted bounding box coordinates and the ground truth bounding box coordinates. This is calculated for each RoI proposal that has been classified as containing an object. The regression loss is typically based on the smooth L1 loss, which is less sensitive to outliers than the traditional L2 loss. The total loss is the sum of the classification loss and the localization loss, weighted by a factor that balances the importance of the two components.

It can be mathematically represented as follows:

$$L_{p,p,t,t} = \; L_{cls}(p,p) + \lambda L_{loc}(t,t) \tag{1}$$

λ is a hyperparameter that balances the importance of the classification and localization losses.

2.5. Layer-wise Learning Rate Fine Tuning

Layer-wise learning rate has been used in all kinds of tasks and models such as BERT [11]. The general idea of layer-wise learning rate is that different layers are capturing different types of information and thus should be fine-tuned to different extents. This paper [12] proposed a similar method for BERT model fine-tuning, the goal is to tweak layers that extract more general features and information less than layers that are more specific to the fine-tuning task. This method was also adopted in other papers like XLNet [13] and ELECTRA [14]. AutoLR [15] is another similar method.

3. Datasets

The key to few-shot learning lies in the generalization ability of the pertinent model when presented with novel categories. Object detection datasets contain many images, each of which is labeled with bounding boxes around objects of interest and the classification of those objects. Thus, it is essential to have a dataset with a wide variety of object categories and high diversity in order to train a base detector that can detect objects and possibly generalize to new ones. In our work, we use two different object detection datasets, one for base training and the other for fine-tuning. A data instance is shown in Figure 2. We follow the few-shot object dataset detection settings in the paper. There are a set of base classes C_b that have many instances and a set of novel classes C_n that have only K (usually less than 10) instances per category. For an object detection dataset $D = \{(x, y), x \in X, y \in Y\}$, where x is the input image and $y = \{(c_i, \mathbf{l}_i), i = 1, ..., N\}$ denotes the categories $c \in C_b \cup C_n$ and bounding box coordinates $\mathbf{l}$ of the N object instances in the image x. Figure 3 shows some samples from the dataset.

3.1. PASCAL VOC

Our base object detector used weights from a model that was previously trained on a subset of classes from the PASCAL VOC dataset provided by previous work.[16][2] This dataset is a commonly used object detection dataset that contains many labeled images of common real-world things such as cats, airplanes, or bottles. PASCAL VOC can also be used for segmentation and classification tasks. For our purposes, we use 20 object categories including animals, vehicles, and household objects, and include around 11,000 images with bounding boxes and classification. Furthermore, PASCAL VOC is very small compared to the COCO dataset, another very popular object detection dataset with a similar scope. It has been a popular benchmark for object detection algorithms. It can be used as a source of pre-training data for object detection models. However, since it has fewer categories and images, it may be less effective for training models that need to recognize a wider range of objects in complex scenes.

3.2. Custom Dataset

Our overarching goal is to create an object detector that, on a small number of labeled examples, can detect Valorant characters from images within the game. For this purpose, we created our own custom dataset containing these examples. The custom dataset is a labeled dataset that we created for few-shot object detection in a video game. It includes 10 images of each character for the training dataset and 10 images

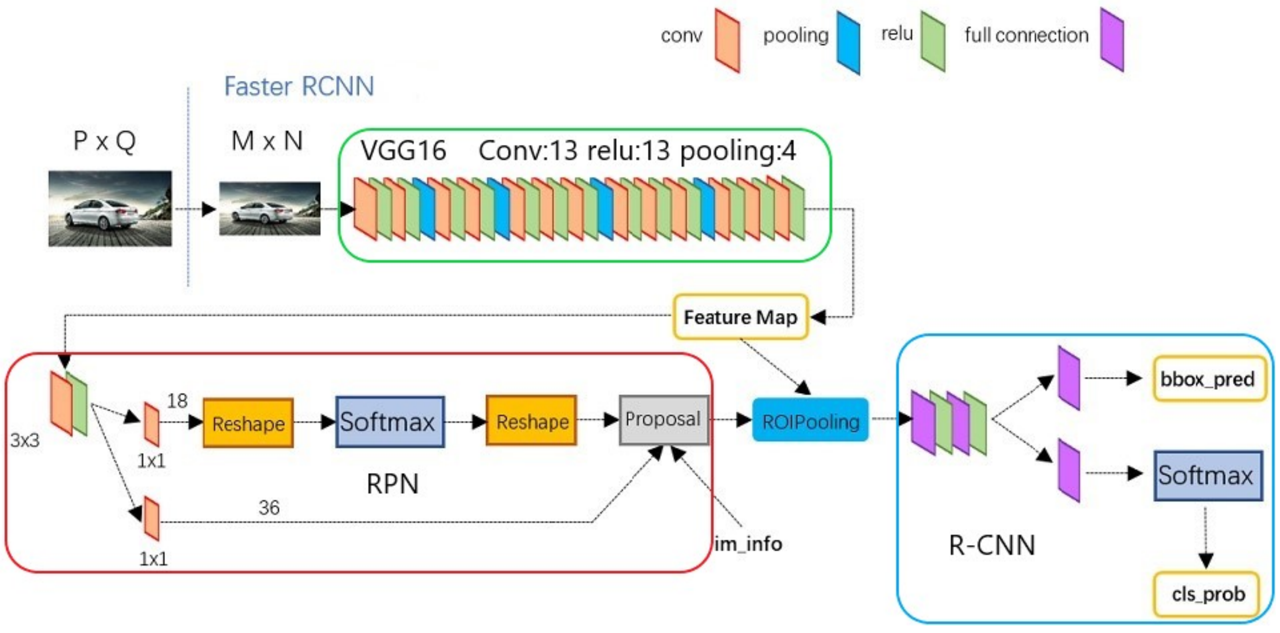

Figure 1: *Faster R-CNN Architecture*

for the validation dataset. These images were screenshots taken in the game with the character of interest in different poses and at different distances from the observer. This labeled dataset is used to fine-tune the pre-trained object detection model. The dataset is publicly available at the following link: https://universe.roboflow.com/fsodvalorant/fsod-valorant

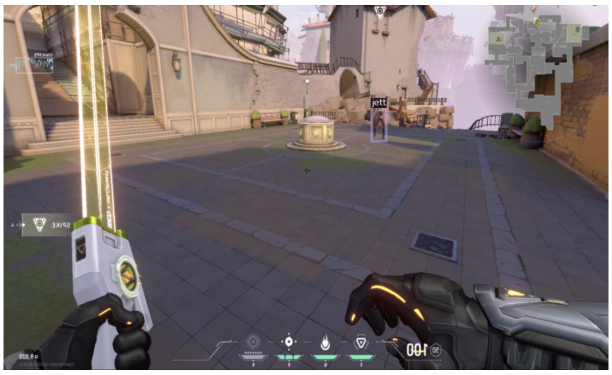

Figure 2: *One data example, the box annotates the name and position of the character*

4. Method

FsDet [2] has a two-stage training scheme as Figure 4 shows. In stage 1, it trained a base object detector on abundant base images and in stage 2 it added some novel pictures, fixed the feature extractor, retrained the final classification layer. As shown in Figure 4, the backbone is a ResNet. We would use ResNet 101 for the project. In the middle part, there is a region proposal network (RPN), as well as a two-layer fully-connected (FC) sub-network as a proposal-level feature extractor. There is also a box predictor composed of a box classifier C to classify the object categories and a box regressor R to predict the

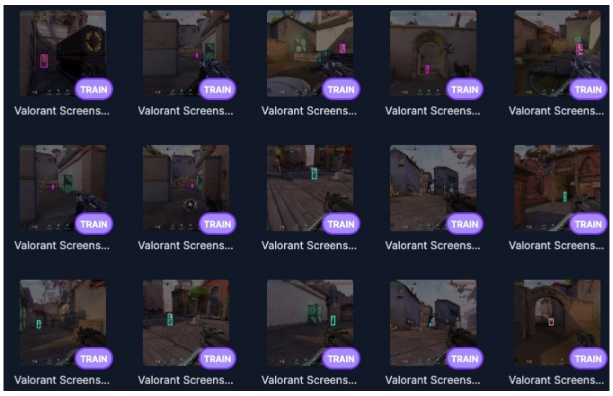

Figure 3: *Our Valorant Dataset Samples*

bounding box coordinates. Intuitively, the backbone features as well as the RPN features are class-agnostic. Therefore, features learned from the base classes are likely to transfer to the novel classes without further parameter updates. [2]

For stage 1, we would use one of the pre-trained models published by the paper named **faster_rcn_R_101_FPN_base1**. The base model we would use is built upon a ResNet-101 backbone network, which is a deep residual network architecture that has been pre-trained on the ImageNet dataset. The FPN is then applied to the output of the ResNet-101 backbone to generate a set of feature maps at different scales. These feature maps are then fed into the RPN, which generates object proposals by predicting the likelihood of an object being present at each location and scale in the feature maps. The object proposals generated by the RPN are then passed through a series of RoI pooling layers, which convert each proposal into a fixed-sized feature map that can be processed by the RCNN. The RCNN consists of a set of fully connected layers that perform object classification and bounding box regression on the RoI features.

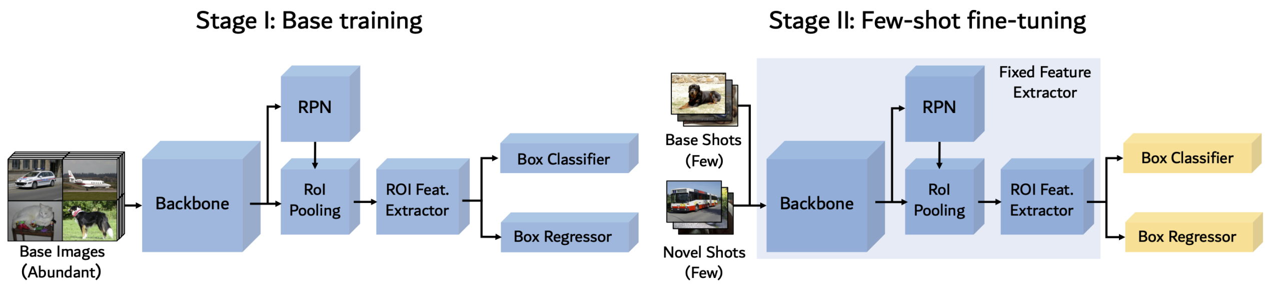

Figure 4: *Original Training and Fine tuning method in FsDet [2]*

In stage 2, the model is fine-tuned. In few-shot fine-tuning. the model would create a small balanced training set with $K = 10$ shots per class. This retrain would often completely remove the classification layer, and reinitialize it with a new size to match novel shots categories. Then the model trains on these few shots. However, the feature extractor does not always perform optimally using a naive training scheme. Thus we proposed Multiplicative Layer-wise Learning Rates methods. This method would allow layers that are close to classification layers (defined as "bottom layers") to be retrained more but partially freeze layers that are close to image inputs (defined as "top layers").

As Figure 5 shows, we start from the bottom layer. The bottom layer marked as $layer_N$ has a basic learning rate lr_{basic}. We introduce another parameter M which denotes that from $layer_M$ to $layer_N$ we apply MLLR. Each layer's learning rate can be calculated as Equation 2 shows for all the layers except RPN parts:

$$lr_k = \begin{cases} \lambda \cdot lr_{k+1}, N - M \leq k \leq N - 1 \\ 0, otherwise \end{cases} \tag{2}$$

Notice that when the learning rate equals 0 or is close to 0, it means that this layer is "frozen" or "almost frozen". Then as the bottom layers would have bigger learning rates and the topper layers have smaller learning rates.

The reason why we do not apply MLLR on RPN part is that it has shortcuts. It is hard to define the layer number in shortcuts, yet the backbone is ResNet, and lower feature extractors and classification layers are all cascading and connected sequentially.

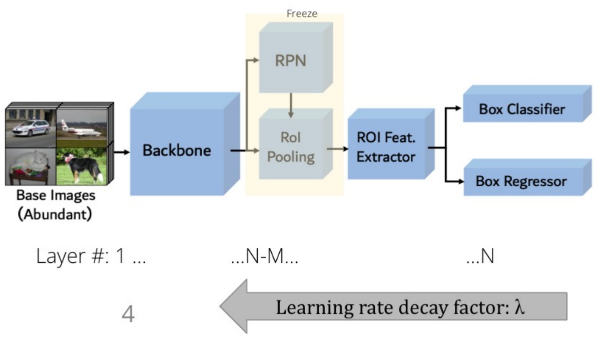

Figure 5: *Multiplicative Layer-wise Learning Rates Fine Tuning, RPN part was frozen as it has shortcuts*

5. Experiments

5.1. Baseline

We have two baseline fine-tuning models.

- Simply following what was posted in the original paper: Freeze all but the last classification layers, as Figure 4 shows. First, assign randomly initialized weights to the box prediction networks for the novel classes and fine-tune only the box classification and regression networks, namely the last layers of the detection model, while keeping the entire feature extractor F fixed. [2]
- Another common fine-tuning method: Unfreeze all layers, remove the last layers, and do retraining on the whole model. This one is just feeding the model with some novel data and novel categories.

5.2. Proposed Method

For our proposed method, we mainly have 3 hyperparameters for ablations. Basic Learning rate lr_{basic} applied on the classification layer, learning rate decay factor $0 \leq \lambda \leq 1$ and retrained layer number $M \leq 118$(The original model structure has a total of 118 layers including RPN part). By setting different values on these hyperparameters and making combinations, we can easily compare and pick the best model for this task.

5.3. Hyperparameter Settings

Each model was trained using a fixed basic learning rate of 0.001 for 3000 iterations. Other hyperparameters vary. Each model costs about 2 hours on a Tesla T4 GPU.

6. Results and Discussions

We evaluated the result on the validation dataset. We use AP and AP50 as evaluation metrics. We tried different values of decay factor and layer numbers and got varying results. Out of the three approaches with annotations shown in Table 1, our training method managed to perform the best by a relatively small margin.

The MLLR model using $\lambda = 0.25, M = 10$ and $\lambda = 0.5, M = 2$, performed very similarly to the model using the Freeze method. This was expected because either not enough layers are retrained($M = 2$) or layers will reach zero learning rate as the decay factor is small($\lambda = 0.25$), effectively reducing the learning rate and freezing layers. The MLLR model using $\lambda = 0.5$ and $\lambda = 0.9$, both performed very well, as they give enough learning rate to topper layers. It appears that lower decay factors generally perform worse, while higher ones perform

Table 1: *AP and AP50 novel categories performance on Valorant dataset of models with different hyperparameters*

Model parameters	**AP**	**AP50**	**Annotations**
Freeze ($\lambda = 1, M = 1$)	18.4	30.1	Baseline for freezing all layers except last classification layers
No-freeze($\lambda = 1, M = 118$)	49.0	75.9	Baseline for unfreezing all layers
$\lambda = 1, M = 50$	47.7	58.8	No LR decay, retraining nearly 1/2 of all layers
$\lambda = 1, M = 75$	48.7	57.4	No LR decay, retraining nearly 3/4 of all layers
$\lambda = 0.25, M = 10$	14.8	26.1	Retraining till the last block of ResNet, RPN excluded
$\lambda = 0.5, M = 2$	14.3	28.7	
$\lambda = 0.5, M = 25$	**51.5**	**81.8**	
$\lambda = 0.5, M = 50$	48.9	81.1	
$\lambda = 0.5, M = 75$	48.5	77.1	
$\lambda = 0.9, M = 25$	49.1	80.8	Retraining nearly 1/4 of all layers
$\lambda = 0.9, M = 50$	**51.9**	**82.4**	
$\lambda = 0.9, M = 75$	50.0	81.5	
$\lambda = 0.9, M = 100$	47.0	76.9	Retraining all most all layers
$\lambda = 0.75, M = 50$	47.5	62.4	

overall better. The sweet spot was found at $\lambda = 0.5, M = 25$ and $\lambda = 0.9, M = 50$.

We guess that the original base model was trained for real-world objects but a game environment is more virtual. The difference between real-world and human-made game settings causes poor performance when applying to retrain of the classification layers. On the one hand, the "Freeze" model fine-tuning is not enough for this game setting and we need to retrain more layers. On the other hand, the "No-freeze" model fine-tuning is too much for this setting as the game mimics the real world and affects the generalization ability. What we did in the "No-freeze" baseline is more likely to feed the model trained on the real-world dataset with "fake" data, resulting in underperformance compared to other models.

We also log diagrams using Wandb and found other interesting things. Our diagram logged the metrics every 200 epochs during training. By fixing M=50 and changing λ, we can find that they follow a similar start but converges differently after 2000 epochs, as shown in 6. A similar outcome for fixing $\lambda = 0.9$ and changing M, as shown in Figure 7. It looks like the hyperparameters will work to different extents as epochs grow.

7. Conclusion

In conclusion, our MLLR training approach has a better performance on the Valorant dataset compared to normal fine-tuning methods. This intuitively makes sense when we interpret freezing all or no layers as setting the decay factors λ to 0 and 1 respectively since it's very possible that the optimal λ is somewhere in between. This may indicate that the practice of freezing layers should be re-evaluated.

8. References

[1] X. L. Yuning Chai, Yongming Rao and X. Wei, "Multiplicative layer-wise learning rates for few-shot object detection," *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[2] X. Wang, T. E. Huang, T. Darrell, J. E. Gonzalez, and F. Yu, "Frustratingly simple few-shot object detection," *arXiv preprint arXiv:2003.06957*, 2020.

[3] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer, 2020, pp. 213–229.

[4] S. Antonelli, D. Avola, L. Cinque, D. Crisostomi, G. L. Foresti, F. Galasso, M. R. Marini, A. Mecca, and D. Pannone, "Few-shot object detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–37, 2022.

[5] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[7] Q. Fan, W. Zhuo, C.-K. Tang, and Y.-W. Tai, "Few-shot object detection with attention-rpn and multi-relation detector," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 4013–4022.

[8] A. Bar, X. Wang, V. Kantorov, C. J. Reed, R. Herzig, G. Chechik, A. Rohrbach, T. Darrell, and A. Globerson, "Detreg: Unsupervised pretraining with region priors for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 14 605–14 615.

[9] T.-Y. Lin, "Coco:common object in context," https://cocodataset.org/#home, May 2023.

[10] e. a. Junchi Yan, "Few-shot object detection with attention-rpn and multi-relation detector," *Conference: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)7*, pp. 13 545–13 553, 2020.

[11] T. Zhang, F. Wu, A. Katiyar, K. Q. Weinberger, and Y. Artzi, "Revisiting few-sample bert fine-tuning," *arXiv preprint arXiv:2006.05987*, 2020.

[12] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[13] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[14] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[15] Y. Ro and J. Y. Choi, "Autolr: Layer-wise pruning and auto-tuning of learning rates in fine-tuning of deep networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 3, 2021, pp. 2486–2494.

[16] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

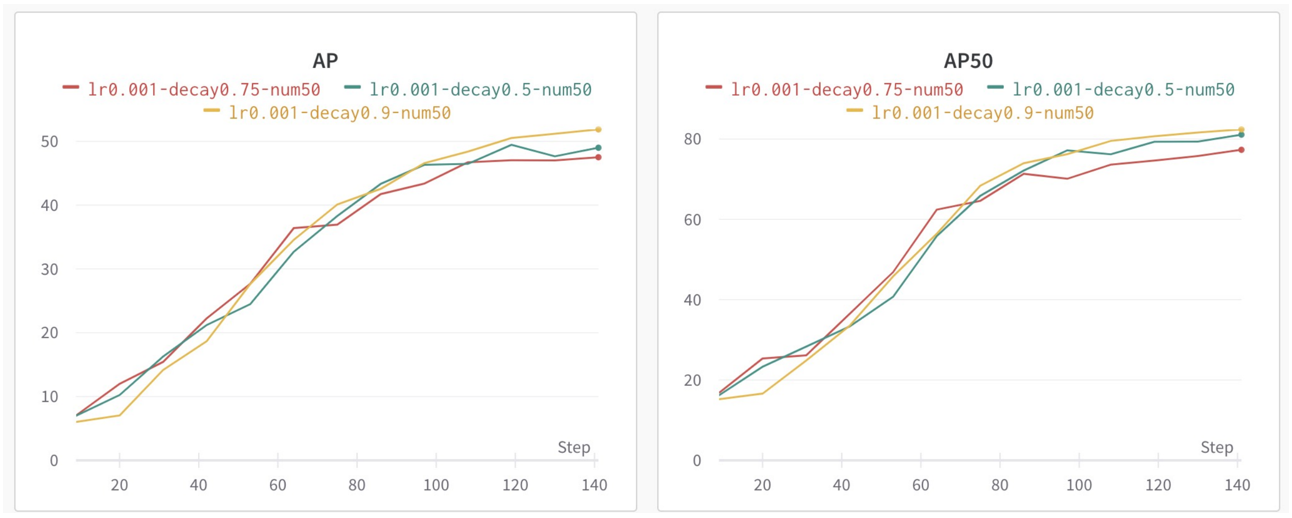

Figure 6: *AP and AP50 performance diagrams for M=50*

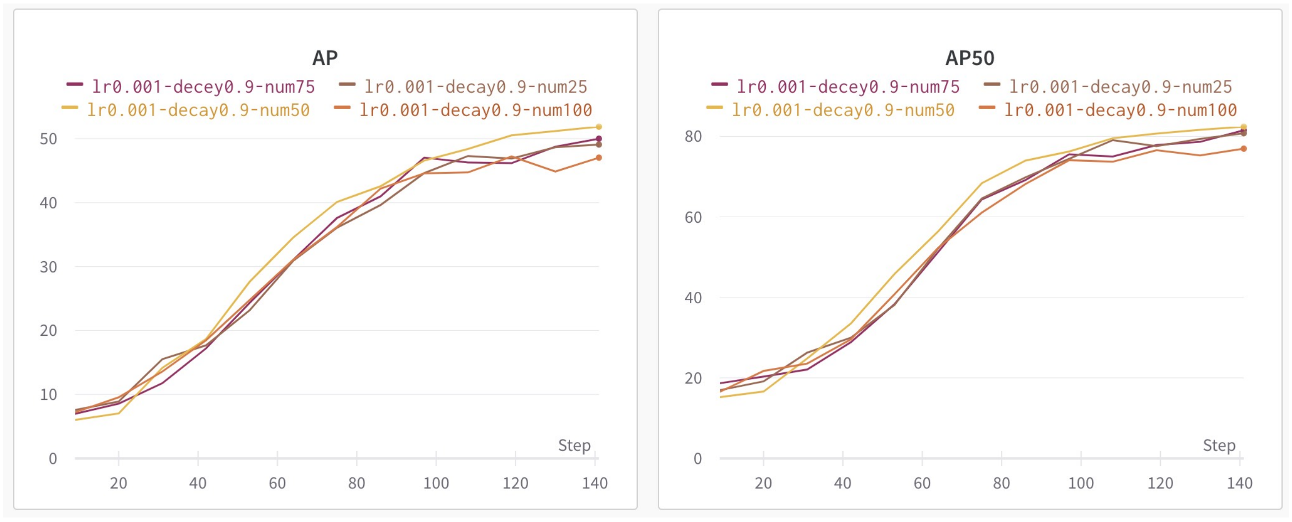

Figure 7: *AP and AP50 performance diagrams for* $\lambda = 0.9$