# MultIOD: Rehearsal-free Multihead Incremental Object Detector

Supplementary Material

## 8. Illustration of Background Interference

Figure 3 illustrates background interference. In the initial state, the model learns correctly how to predict bicycles. In the incremental state, the bicycle is not anymore annotated, which causes: (1) background shift as the bicycle is confused with background, and (2) catastrophic forgetting because of distribution shift towards the new class. Here, the model learns correctly cars but fails to detect bicycles.

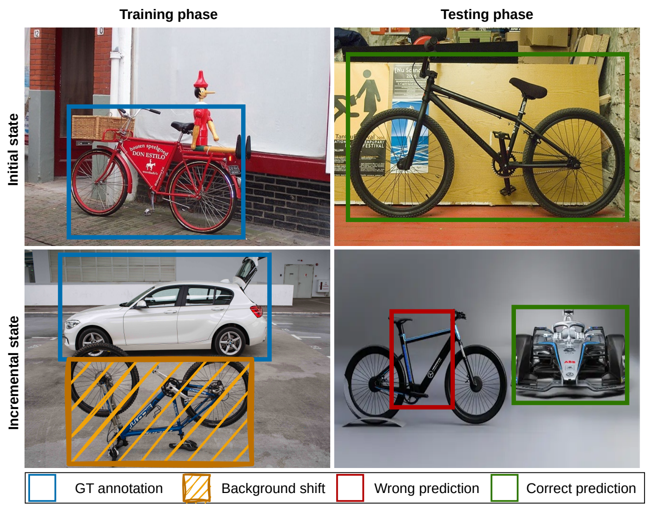

Figure 3. Illustration of background interference

## 9. Implementation Details

We implement our method using Keras (Tensorflow 2.8.0). We use Adam optimizer with a learning rate of $2e^{-4}$, a batch size of 16, and a weight decay of $1.25e-5$. Similarly to [51], we use images of size $512 \times 512$, down-sampled 4 times to have prediction maps of size $128 \times 128$. All our models and those of compared methods are pretrained with imagenet weights. We use a detection threshold of 5% to compute the mean-average-precision (mAP), even thought we could use lower threshold to improve results, we prefer to keep the model inference time bounded (in the state of the art, a value of 1% is usually used).

For VOC datasets [11], we train our model for 70 epochs in each state, we decay the learning rate by 10 at epochs 45 and 60. For training, we use as augmentations random flip, random resized crop, color jittering, and random scale. For testing, we use flip augmentation like in [12, 34, 51]. As mentioned in the main paper, the classes of VOC are ordered alphabetically before being divided into groups. Figure 4 shows this order and reminds the protocol used.

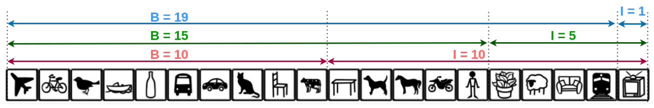

Figure 4. Pascal VOC incremental protocol

For MNIST dataset [20], we train our model for 20 epochs in each state, and keep the other hyper-parameters unchanged. For training, we use as augmentations random resized crop and random scale. For testing, we use flip augmentation.

## 10. Feature Pyramids Architecture

In $MultIOD$, each feature pyramid is constructed of 4 levels that are connected using dropout layers to the backbone. The connected layers of backbone are colored in light gray in Figure 5, and are specified for each EfficientNet variant in Table 7. Layer names given in this table are based on the official $keras - applications$ implementations.

As shown in Figure 5, each feature pyramid contains three blocks of layers each containing: upsampling $2 \times 2$, convolution layer with number of filters shown between parenthesis, batch normalization layer and ReLU, concatenation layer, another convolutional layer, batch norm and ReLU. Upsampling is done progressively in order to capture multi-scale features. We use the FPN implementation of this GitHub repository [2]. In the class-wise feature pyramids (Subsection 5.2 of the main paper), we use the same architecture described in Figure 5, but we reduce the number of filters in the convolutional layers to avoid an explosion in the number of parameters. We thus use only 64, 64, 32, 32, 16, and 16 filters in each convolutional layer, respectively.

## 11. MNIST Dataset Creation Details

We made sure to create a challenging dataset by doing the following:

- We set the minimum and maximum digit sizes between $50^2$ and $200^2$ pixels, respectively, in order to have both small and large digits.
- We make sure to have one to five digits in each image, for diversification.
- The background shift is present in this dataset as we randomly pick digits from a set of ten, regardless of the current state.

Examples of generated images are in Figure 6.

[2]https://github.com/Ximilar-com/xcenternet

| Backbone | Level 1 | Level 2 | Level 3 | Level 4 |
|---|---|---|---|---|
| EfficientNet-B0 | block2b-activation | block3b-activation | block5c-activation | top-activation |
| EfficientNet-B3 | block2c-activation | block3c-activation | block5e-activation | top-activation |
| EfficientNet-B5 | block2e-activation | block3e-activation | block5g-activation | top-activation |

Table 7. Names of layers in Keras corresponding to Feature Pyramid [25] Levels for different EfficientNet architectures

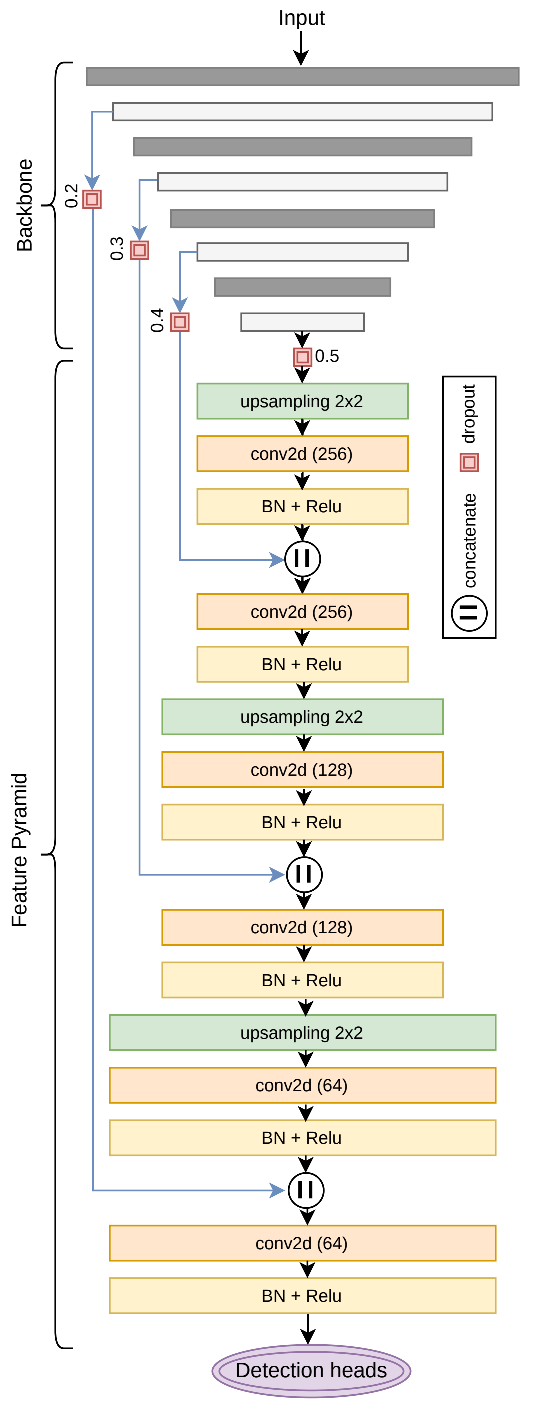

Figure 5. Architecture of one Feature Pyramid in $MultIOD$

## 12. Results with mAP@[0.5, 0.95]

Tables 8 and 9 provide results of our method on VOC2007 and VOC0712, using mAP averaged over IoU threshold that varies between 0.5 and 0.95 with a step of 0.05. Results are provided for future comparisons.

| Method | Full | $B = 19, I = 1$ | | $B = 15, I = 5$ | | $B = 10, I = 10$ | |
|---|---|---|---|---|---|---|---|
| | $mAP$ | $mAP$ | $F_{mAP}$ | $mAP$ | $F_{mAP}$ | $mAP$ | $F_{mAP}$ |
| IoU = 0.5 | 60.4 | 59.9 | 39.2 | 52.6 | 37.8 | 48.4 | 47.2 |
| IoU = [0.5, 0.95] | 35.9 | 35.7 | 18.5 | 30.7 | 20.2 | 25.9 | 23.9 |

Table 8. Mean-average-precision and $F_{mAP}$ score on VOC2007.

| Method | Full | $B = 19, I = 1$ | | $B = 15, I = 5$ | | $B = 10, I = 10$ | |
|---|---|---|---|---|---|---|---|
| | $mAP$ | $mAP$ | $F_{mAP}$ | $mAP$ | $F_{mAP}$ | $mAP$ | $F_{mAP}$ |
| IoU = 0.5 | 69.5 | 68.0 | 56.9 | 60.7 | 47.0 | 56.6 | 55.8 |
| IoU = [0.5, 0.95] | 45.7 | 44.5 | 33.6 | 39.0 | 26.8 | 33.2 | 31.0 |

Table 9. Mean-average-precision and $F_{mAP}$ score on VOC0712.

## 13. Ablation of Backbones on MNIST

In Table 10, we provide results of $MultIOD$ using different backbones on MNIST dataset. Because this dataset is not challenging and is of a small size, it is easier for large models like EfficientNet [41] to learn it. In our experiments, it is hard to determine which backbone provides the best results for this dataset, as each backbone is best in one configuration. However, results of different models are comparable, and we thus recommend using the smallest version (EfficientNet-B0) for this dataset.

| Method | Full | $B = 9, I = 1$ | | $B = 7, I = 3$ | | $B = 5, I = 5$ | |
|---|---|---|---|---|---|---|---|
| | $mAP$ | $mAP$ | $F_{mAP}$ | $mAP$ | $F_{mAP}$ | $mAP$ | $F_{mAP}$ |
| EfficientNet-B0 | 93.1 | 91.3 | 91.2 | **93.1** | **93.5** | 91.3 | 91.3 |
| EfficientNet-B3 | 91.1 | **91.7** | **92.6** | 92.0 | 92.4 | 89.7 | 89.7 |
| EfficientNet-B5 | **93.7** | 91.2 | 92.4 | 90.6 | 91.4 | **92.5** | **92.5** |

Table 10. Ablation of backbones on MNIST dataset (mAP@0.5).

## 14. Ablation of NMS Strategies on VOC2007

In Table 11, we provide the results of $MultIOD$ using different NMS strategies on VOC2007 dataset. Similarly to the results presented in the main paper, the method that achieves the best results is class-wise NMS, followed by inter-class NMS. Soft-NMS and No-NMS are the methods that achieve the lowest results.

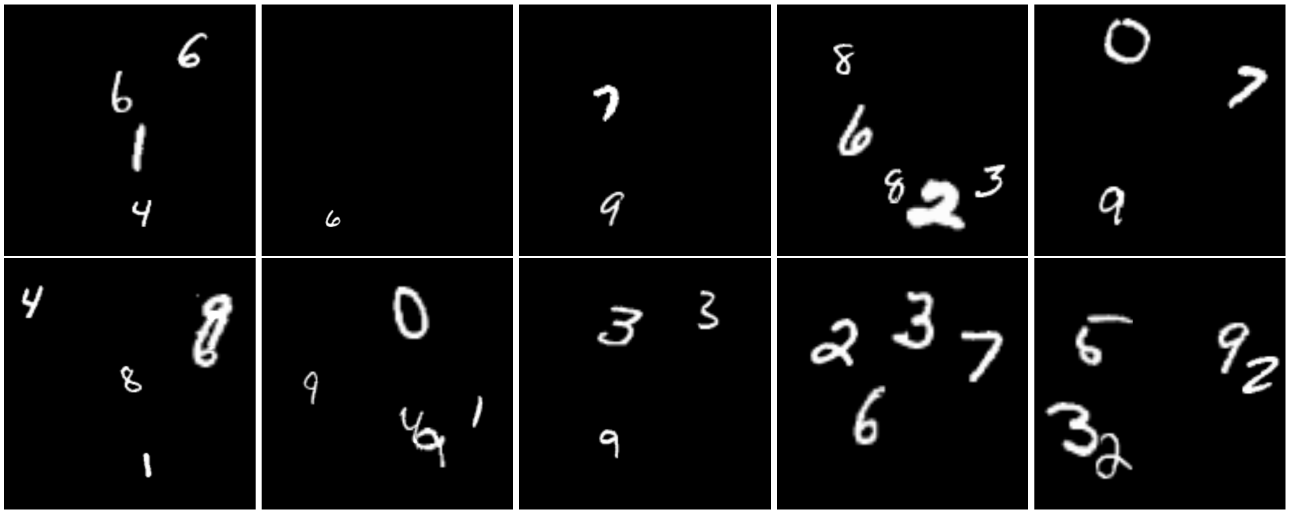

Figure 6. Examples of generated MNIST images

| Method | Full | $B=19, I=1$ | | $B=15, I=5$ | | $B=10, I=10$ | |
|---|---|---|---|---|---|---|---|
| | $mAP$ | $mAP$ | $F_{mAP}$ | $mAP$ | $F_{mAP}$ | $mAP$ | $F_{mAP}$ |
| No-NMS | 51.7 | 51.6 | 33.3 | 44.4 | 28.7 | 36.9 | 33.2 |
| Soft-NMS | 45.8 | 46.6 | 29.6 | 40.5 | 23.8 | 34.5 | 31.4 |
| Inter-class NMS | 53.0 | 51.8 | 35.7 | 46.1 | 34.1 | 41.9 | 40.1 |
| Class-wise NMS | **56.7** | **55.7** | **35.6** | **49.2** | **33.8** | **46.3** | **45.5** |

Table 11. Performance of our model using VOC2007 dataset with different NMS strategies and EfficientNet-B0.

## 15. Examples of Detections with MultIOD

Figure 7 provides examples of predictions made with our $MultIOD$ continual detector. Orange is used for past class detections, and blue is used for new class detections. Visual results confirm the robustness of our method against catastrophic forgetting. $MultIOD$ provides a good compromise between stability of the neural network and its plasticity.

## 16. Comparison Against Two-Stage Detectors

Table 7 provides a comparison of $MultIOD$ with some two-stage continual detectors on VOC2007 dataset. Rehearsal-based methods store a subset of past data, and replay it when training new classes to tackle catastrophic forgetting.

| Method | Detector | Rehearsal? | $B=19, I=1$ | $B=15, I=5$ | $B=10, I=10$ |
|---|---|---|---|---|---|
| MultIOD | CenterNet | × | 59.9 | 52.6 | 48.4 |
| MVD [46] | Faster R-CNN | × | 69.7 | 66.5 | 66.1 |
| IncDet [26] | Fast(er) R-CNN | × | × | 70.4 | 70.8 |
| RD-IOD [45] | Faster R-CNN | × | 72.1 | 69.7 | 66.2 |
| Faster-ILOD [33] | Faster R-CNN | × | 68.6 | 68.0 | 62.2 |
| ORE [18] | Faster R-CNN | ✓ | 68.9 | 68.5 | 64.6 |
| OST [47] | Faster R-CNN | ✓ | 69.8 | 69.9 | 65.0 |

Table 12. mAP@0.5 scores on VOC2007 dataset.

Results indicate that $MultIOD$ achieves the lowest results compared to methods that combine both two-stage detectors and rehearsal memory. This is intuitive because with the absence of memory from the past, inter-class separability becomes more challenging.

Fast(er)-RCNN are two-stage detectors that perform better than CenterNet, but are very slow which make them not suitable for real-life applications. A trade-off is required to select between the two detectors depending on the use case.

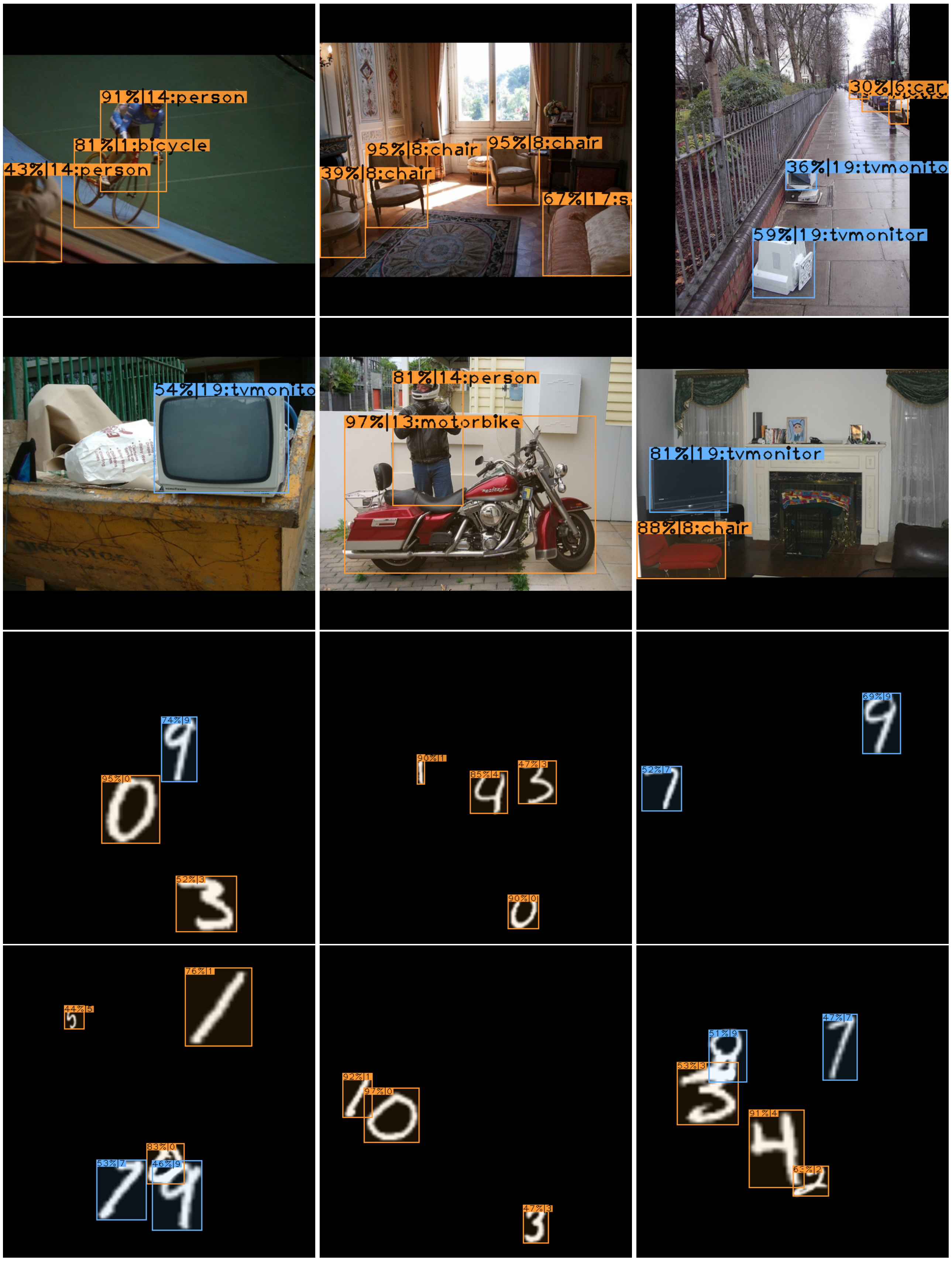

Figure 7. Examples of detections with $MultIOD$ on VOC0712 (EfficientNet-B3, B=19, I=1) and MNIST (EfficientNet-B0, B=7, I=3)