

---

## Faster R-CNN and Mask R-CNN in PyTorch 1.0

**maskrcnn-benchmark has been deprecated. Please see detectron2, which includes implementations for all models in maskrcnn-benchmark**

This project aims at providing the necessary building blocks for easily creating detection and segmentation models using PyTorch 1.0.



### Highlights

- **PyTorch 1.0:** RPN, Faster R-CNN and Mask R-CNN implementations that matches or exceeds Detectron accuracies
- **Very fast:** up to **2x** faster than Detectron and **30%** faster than mmdetection during training. See MODEL\_ZOO.md for more details.
- **Memory efficient:** uses roughly 500MB less GPU memory than mmdetection during training
- **Multi-GPU training and inference**
- **Mixed precision training:** trains faster with less GPU memory on NVIDIA tensor cores.

- 
- **Batched inference:** can perform inference using multiple images per batch per GPU
  - **CPU support for inference:** runs on CPU in inference time. See our webcam demo for an example
  - Provides pre-trained models for almost all reference Mask R-CNN and Faster R-CNN configurations with 1x schedule.

## Webcam and Jupyter notebook demo

We provide a simple webcam demo that illustrates how you can use [maskrcnn\\_benchmark](#) for inference:

```
1 cd demo
2 # by default, it runs on the GPU
3 # for best results, use min-image-size 800
4 python webcam.py --min-image-size 800
5 # can also run it on the CPU
6 python webcam.py --min-image-size 300 MODEL.DEVICE cpu
7 # or change the model that you want to use
8 python webcam.py --config-file ../configs/caffe2/
   e2e_mask_rcnn_R_101_FPN_1x_caffe2.yaml --min-image-size 300 MODEL.
   DEVICE cpu
9 # in order to see the probability heatmaps, pass --show-mask-heatmaps
10 python webcam.py --min-image-size 300 --show-mask-heatmaps MODEL.DEVICE
   cpu
11 # for the keypoint demo
12 python webcam.py --config-file ../configs/caffe2/
   e2e_keypoint_rcnn_R_50_FPN_1x_caffe2.yaml --min-image-size 300 MODEL
   .DEVICE cpu
```

A notebook with the demo can be found in `demo/Mask_R-CNN_demo.ipynb`.

## Installation

Check `INSTALL.md` for installation instructions.

## Model Zoo and Baselines

Pre-trained models, baselines and comparison with Detectron and mmdetection can be found in `MODEL_ZOO.md`

---

## Inference in a few lines

We provide a helper class to simplify writing inference pipelines using pre-trained models. Here is how we would do it. Run this from the `demo` folder:

```
1 from maskrcnn_benchmark.config import cfg
2 from predictor import COCODemo
3
4 config_file = "../configs/caffe2/e2e_mask_rcnn_R_50_FPN_1x_caffe2.yaml"
5
6 # update the config options with the config file
7 cfg.merge_from_file(config_file)
8 # manual override some options
9 cfg.merge_from_list(["MODEL.DEVICE", "cpu"])
10
11 coco_demo = COCODemo(
12     cfg,
13     min_image_size=800,
14     confidence_threshold=0.7,
15 )
16 # load image and then run prediction
17 image = ...
18 predictions = coco_demo.run_on_opencv_image(image)
```

## Perform training on COCO dataset

For the following examples to work, you need to first install `maskrcnn_benchmark`.

You will also need to download the COCO dataset. We recommend to symlink the path to the coco dataset to `datasets/` as follows

We use `minival` and `valminusminival` sets from Detectron

```
1 # symlink the coco dataset
2 cd ~/github/maskrcnn-benchmark
3 mkdir -p datasets/coco
4 ln -s /path_to_coco_dataset/annotations datasets/coco/annotations
5 ln -s /path_to_coco_dataset/train2014 datasets/coco/train2014
6 ln -s /path_to_coco_dataset/test2014 datasets/coco/test2014
7 ln -s /path_to_coco_dataset/val2014 datasets/coco/val2014
8 # or use COCO 2017 version
9 ln -s /path_to_coco_dataset/annotations datasets/coco/annotations
10 ln -s /path_to_coco_dataset/train2017 datasets/coco/train2017
11 ln -s /path_to_coco_dataset/test2017 datasets/coco/test2017
12 ln -s /path_to_coco_dataset/val2017 datasets/coco/val2017
13
14 # for pascal voc dataset:
15 ln -s /path_to_VOCdevkit_dir datasets/voc
```

---

P.S. `COCO_2017_train = COCO_2014_train + valminusminival` , `COCO_2017_val = minival`

You can also configure your own paths to the datasets. For that, all you need to do is to modify `maskrcnn_benchmark/config/paths_catalog.py` to point to the location where your dataset is stored. You can also create a new `paths_catalog.py` file which implements the same two classes, and pass it as a config argument `PATHS_CATALOG` during training.

## Single GPU training

Most of the configuration files that we provide assume that we are running on 8 GPUs. In order to be able to run it on fewer GPUs, there are a few possibilities:

### 1. Run the following without modifications

```
1 python /path_to_maskrcnn_benchmark/tools/train_net.py --config-file "/  
   path/to/config/file.yaml"
```

This should work out of the box and is very similar to what we should do for multi-GPU training. But the drawback is that it will use much more GPU memory. The reason is that we set in the configuration files a global batch size that is divided over the number of GPUs. So if we only have a single GPU, this means that the batch size for that GPU will be 8x larger, which might lead to out-of-memory errors.

If you have a lot of memory available, this is the easiest solution.

### 2. Modify the cfg parameters

If you experience out-of-memory errors, you can reduce the global batch size. But this means that you'll also need to change the learning rate, the number of iterations and the learning rate schedule.

Here is an example for Mask R-CNN R-50 FPN with the 1x schedule:

```
1 python tools/train_net.py --config-file "configs/  
   e2e_mask_rcnn_R_50_FPN_1x.yaml" SOLVER.IMS_PER_BATCH 2 SOLVER.  
   BASE_LR 0.0025 SOLVER.MAX_ITER 720000 SOLVER.STEPS "(480000, 640000)"  
   " TEST.IMS_PER_BATCH 1 MODEL.RPN.FPN_POST_NMS_TOP_N_TRAIN 2000"
```

This follows the scheduling rules from Detectron. Note that we have multiplied the number of iterations by 8x (as well as the learning rate schedules), and we have divided the learning rate by 8x.

We also changed the batch size during testing, but that is generally not necessary because testing requires much less memory than training.

Furthermore, we set `MODEL.RPN.FPN_POST_NMS_TOP_N_TRAIN 2000` as the proposals are selected for per the batch rather than per image in the default training. The value is calculated by **1000 x images-per-gpu**. Here we have 2 images per GPU, therefore we set the number as  $1000 \times 2 = 2000$ .

---

If we have 8 images per GPU, the value should be set as 8000. Note that this does not apply if `MODEL.RPN.FPN_POST_NMS_PER_BATCH` is set to `False` during training. See #672 for more details.

## Multi-GPU training

We use internally `torch.distributed.launch` in order to launch multi-gpu training. This utility function from PyTorch spawns as many Python processes as the number of GPUs we want to use, and each Python process will only use a single GPU.

```
1 export NGPUS=8
2 python -m torch.distributed.launch --nproc_per_node=$NGPUS /
  path_to_maskrcnn_benchmark/tools/train_net.py --config-file "path/to
  /config/file.yaml" MODEL.RPN.FPN_POST_NMS_TOP_N_TRAIN images_per_gpu
  x 1000
```

Note we should set `MODEL.RPN.FPN_POST_NMS_TOP_N_TRAIN` follow the rule in Single-GPU training.

## Mixed precision training

We currently use APEX to add Automatic Mixed Precision support. To enable, just do Single-GPU or Multi-GPU training and set `DTYPE "float16"`.

```
1 export NGPUS=8
2 python -m torch.distributed.launch --nproc_per_node=$NGPUS /
  path_to_maskrcnn_benchmark/tools/train_net.py --config-file "path/to
  /config/file.yaml" MODEL.RPN.FPN_POST_NMS_TOP_N_TRAIN images_per_gpu
  x 1000 DTYPE "float16"
```

If you want more verbose logging, set `AMP_VERBOSE True`. See Mixed Precision Training guide for more details.

## Evaluation

You can test your model directly on single or multiple gpus. Here is an example for Mask R-CNN R-50 FPN with the 1x schedule on 8 GPUS:

```
1 export NGPUS=8
2 python -m torch.distributed.launch --nproc_per_node=$NGPUS /
  path_to_maskrcnn_benchmark/tools/test_net.py --config-file "configs/
  e2e_mask_rcnn_R_50_FPN_1x.yaml" TEST.IMS_PER_BATCH 16
```

---

To calculate mAP for each class, you can simply modify a few lines in `coco_eval.py`. See #524 for more details.

## Abstractions

For more information on some of the main abstractions in our implementation, see `ABSTRACTIONS.md`.

## Adding your own dataset

This implementation adds support for COCO-style datasets. But adding support for training on a new dataset can be done as follows:

```
1 from maskrcnn_benchmark.structures.bounding_box import BoxList
2
3 class MyDataset(object):
4     def __init__(self, ...):
5         # as you would do normally
6
7     def __getitem__(self, idx):
8         # load the image as a PIL Image
9         image = ...
10
11         # load the bounding boxes as a list of list of boxes
12         # in this case, for illustrative purposes, we use
13         # x1, y1, x2, y2 order.
14         boxes = [[0, 0, 10, 10], [10, 20, 50, 50]]
15         # and labels
16         labels = torch.tensor([10, 20])
17
18         # create a BoxList from the boxes
19         boxlist = BoxList(boxes, image.size, mode="xyxy")
20         # add the labels to the boxlist
21         boxlist.add_field("labels", labels)
22
23         if self.transforms:
24             image, boxlist = self.transforms(image, boxlist)
25
26         # return the image, the boxlist and the idx in your dataset
27         return image, boxlist, idx
28
29     def get_img_info(self, idx):
30         # get img_height and img_width. This is used if
31         # we want to split the batches according to the aspect ratio
32         # of the image, as it can be more efficient than loading the
33         # image from disk
34         return {"height": img_height, "width": img_width}
```

---

That's it. You can also add extra fields to the boxlist, such as segmentation masks (using `structures.segmentation_mask.SegmentationMask`), or even your own instance type.

For a full example of how the `COCODataset` is implemented, check `maskrcnn_benchmark/data/datasets/coco.py`.

Once you have created your dataset, it needs to be added in a couple of places: -`maskrcnn_benchmark/data/datasets/__init__.py`: add it to `__all__` - `maskrcnn_benchmark/config/paths_catalog.py`: `DatasetCatalog.DATASETS` and corresponding `if` clause in `DatasetCatalog.get()`

## Testing

While the aforementioned example should work for training, we leverage the `cocoApi` for computing the accuracies during testing. Thus, test datasets should currently follow the `cocoApi` for now.

To enable your dataset for testing, add a corresponding `if` statement in `maskrcnn_benchmark/data/datasets/evaluation/__init__.py`:

```
1 if isinstance(dataset, datasets.MyDataset):
2     return coco_evaluation(**args)
```

## Finetuning from Detectron weights on custom datasets

Create a script `tools/trim_detectron_model.py` like here. You can decide which keys to be removed and which keys to be kept by modifying the script.

Then you can simply point the converted model path in the config file by changing `MODEL.WEIGHT`.

For further information, please refer to #15.

## Troubleshooting

If you have issues running or compiling this code, we have compiled a list of common issues in `TROUBLESHOOTING.md`. If your issue is not present there, please feel free to open a new issue.

---

## Citations

Please consider citing this project in your publications if it helps your research. The following is a BibTeX reference. The BibTeX entry requires the `url` LaTeX package.

```
1 @misc{massa2018mrcnn,  
2   author = {Massa, Francisco and Girshick, Ross},  
3   title = {{maskrcnn-benchmark: Fast, modular reference implementation of  
4     Instance Segmentation and Object Detection algorithms in PyTorch}},  
5   year = {2018},  
6   howpublished = {\url{https://github.com/facebookresearch/maskrcnn-  
7     benchmark}},  
8   note = {Accessed: [Insert date here] }  
9 }
```

## Projects using maskrcnn-benchmark

- RetinaMask: Learning to predict masks improves state-of-the-art single-shot detection for free. Cheng-Yang Fu, Mykhailo Shvets, and Alexander C. Berg. Tech report, arXiv,1901.03353.
- FCOS: Fully Convolutional One-Stage Object Detection. Zhi Tian, Chunhua Shen, Hao Chen and Tong He. Tech report, arXiv,1904.01355. [code]
- MULAN: Multitask Universal Lesion Analysis Network for Joint Lesion Detection, Tagging, and Segmentation. Ke Yan, Youbao Tang, Yifan Peng, Veit Sandfort, Mohammadhadi Bagheri, Zhiyong Lu, and Ronald M. Summers. MICCAI 2019. [code]
- Is Sampling Heuristics Necessary in Training Deep Object Detectors? Joya Chen, Dong Liu, Tong Xu, Shilong Zhang, Shiwei Wu, Bin Luo, Xuezheng Peng, Enhong Chen. Tech report, arXiv,1909.04868. [code]

## License

maskrcnn-benchmark is released under the MIT license. See LICENSE for additional details.