BMI / CS 771 Fall 2023:
Homework Assignment 3

Oct 2023

1 Overview

One of the central problems in computer vision is to recognize and localize object instances in an input image. This problem, widely known as object detection, often requires orchestrated model design and implementation. In this assignment, you will implement FCOS — a fully convolutional one-stage object detector, using PyTorch and with our helper code. Further, you will train and evaluate FCOS on PASCAL VOC 2007 [1] dataset using your implementation. The conference version of FCOS [4] is listed as our reading material. This assignment will follow its journal version [5], which incorporates better design choices and provides more details. While the model is conceptually simple, the implementation requires some careful thoughts.

This assignment is team-based and requires cloud computing. A team can have up to 3 students. The assignment has a total of 12 points with unlimited bonus points. Details and rubric are described in Section 4.

2 Setup

- We recommend using Conda to manage your packages.
- The following packages are needed: PyTorch ($\geq$1.11 with GPU support), torchvision, pycocotools, PyYaml, NumPy, Pillow, and Tensorboard. Again, you are in charge of installing them.
- You can install any common packages that are helpful for the implementation. If a new package is required for your implementation, a description must be included in the writeup.
- **Using an open source implementation of FCOS is not permitted in this assignment**, including the one in torchvision. Feel free to take some inspiration from an existing implementation.
- You can debug your code and run experiments on CPUs. However, training a neural network is very expensive on CPUs. GPU computing is thus

required for this project. Please setup your team's cloud instance. **Do remember to shut down the instance when it is not used!**

- You will need PASCAL VOC 2007 dataset for this assignment. We provide a bash script to download this dataset. Simply run
  sh ./download_dataset.sh
- To complete the assignment, you will need to fill in the missing code in:
  ./code/libs/model.py
- You are allowed to modify any part of the code in order to improve the model design or the training / inference of the model, **except the evaluation code**, including *./code/eval.py* and the *evaluate* function in *./code/libs/engine.py*.
- Your submission should include the code, results, a trained model (see Sec. 6) and a writeup. The submission can be generated using:
  python ./zip_submission.py

3 Overview of the Implementation

Our helper code and the implementation (located under ./code) are significantly more complicated than the first two homework assignments. Here we include an overview of the FCOS model and our implementation. You are required to go through our helper code before attempting the implementation.

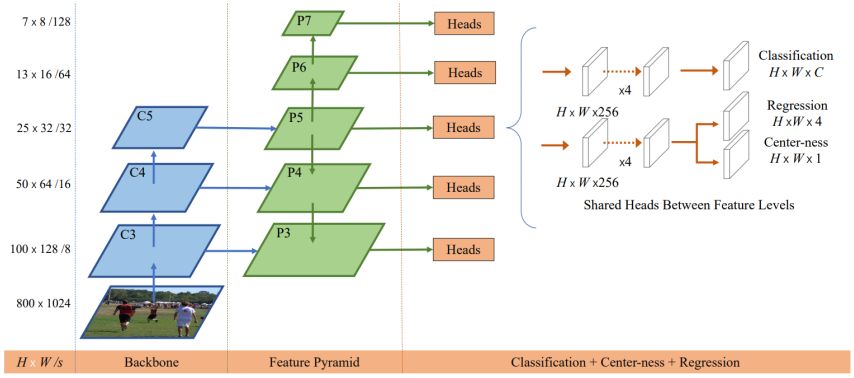

Figure 1: Overview of FCOS from [5]. The model consists of a backbone network, a feature pyramid, and shared classification / regression heads.

The FCOS Model. FCOS, as the name suggested, provides a one-stage object detector. The key idea is to decode an object candidate (including its categorical label and bounding box) at every spot on the feature maps, where the box is represented as its center (located at the spot) plus offsets to four borders.

In doing so, object detection is simplified as a dense image labeling problem. Specifically, the model design, as shown in Figure 1, consists of (a) a backbone (e.g., ResNet [2]) to extract a pyramid of image features; (b) a feature pyramid network [3] (FPN) to refine these image features; and (c) shared classification and regression heads that are applied to the refined feature maps for decoding object labels and boxes.

Code Organization. The helper code are organized into three parts.

- Source code for input pipeline, model architecture, and training/inference utilities are located under ./code/libs/.
- Configuration files (in yaml format) for the model architecture, training, and inference are stored under ./code/configs/.
- Training and evaluation interfaces are exposed in ./code/train.py and ./code/eval.py.

Input Pipeline (./code/libs/dataset.py and ./code/libs/transforms.py). Unlike image classification, labels for object detection (e.g., box coordinates) depend on the location of the image region. Hence, any data augmentation that changes the pixel location (say flipping or resizing an input image) will have to modify the box labels accordingly. Further, input images in the same mini-batch might have different aspect ratios, and thus batching these images can be challenging.

Our helper code has provided a standard input pipeline for training object detection models. The dataset class (*VOCDetection*) provides a wrapper to load and process the VOC dataset. The *GeneralizedRCNNTransform* class implements common data augmentations for object detection, including horizaontal flipping, scale jitters. Proper padding is also included for these augmentations, so that their outputs are of a common size and thus can be easily batched.

Model Architectures (./code/libs/model.py). This is the implementation of the FCOS model, including the backbone, FPN, and heads. The classification and regression heads are implemented in *FCOSClassificationHead* and *FCOSRegressionHead*, while all other parts are encapsulated in the *FCOS* class. The current implementation (./libs/model.py) misses several critical pieces in order to function properly. **You will need to complete the code here.**

Training, Inference, and Other Utilities (all other files under ./code/libs). These files implement utility functions that are necessary for the assignment, including training and inference pipelines, loss functions, visualizations, and others. Two of the key components worth noting are (a) the training and inference pipeline (*./code/libs/engine.py*); and (b) the centralized parameter configuration (*./libs/config.py*). Specifically, engine.py includes functions to train the model, and to evaluate a trained-model. config.py specifies model architecture, model parameters, as well as hyperparameters used in training, which can be overwritten using a external configuration file (in yaml format e.g., ./code/configs/voc_fcos.yaml).

To modify the model architecture or training hyperparameters, we can now replace the corresponding fields and their values in a separate yaml config file, without the need to touch the code. **See ./code/configs/voc_fcos.yaml for an example.** This design allows us to separate the implementation and configuration, and is particularly helpful for the experiments.

Training and Evaluation Interface. Our helper code provides the full training and inference interface as follows (assuming ./code as the current directory).

- To train the model, run
  python ./train.py ./config/voc_fcos.yaml

- By default, the training logs and checkpoints will be saved under a folder in *../logs*. Each run will have a separate folder that ends with the starting time of the experiment. You can monitor and visualize these training logs using TensorBoard. Similar to our previous assignment, we recommend copying the logs and plotting the curves locally.
  tensorboard --logdir=../logs/your_exp_folder

- To evaluate a trained model, run
  python ./eval.py ./config/voc_fcos.yaml ../logs/your_exp_folder

- This evaluation will automatically grab the last checkpoint and save all results (e.g., *eval_results.json* and ./viz folder for visualization) under the experiment folder. Additionally, you can specify which checkpoint to evaluate (using "-e") or visualize the detection results (using "-v"). See eval.py for more details.

4 Details

This assignment has three parts. An autograder will be used to grade certain parts of the assignment. **Please follow the instructions closely.**

4.1 Understanding FCOS

The core idea of FCOS is simple and its results are impressive. Proper implementation of FCOS, however, requires us to dig into its technical details. Before the actual implementation, our first step is to understand the model design of FCOS, especially in the context of our helper code.

Model Design (4 pts). Please read the journal paper [5], go through our helper code, and address the following questions.

- What is the output of the backbone?

- How many levels are considered in FPN? And how does the FPN generates its output feature maps?

- How does FCOS assign positive / negative samples during training? What are the loss functions used in the training?
- How does FCOS decode objects at inference? What are the necessary post-processing steps (e.g., non-maximum suppression)?

4.2 Model Inference

Our first step is to implement the inference part of FCOS.

Classification and Regression Heads (2 pts). You will need to complete the implementation of the classification and regression heads in the FOCS model. Specifically, our helper code provides definitions of the heads in the class of *FCOSClassificationHead* and *FCOSRegressionHead* under ./code/libs/model.py. You will need to implement the forward pass of these two classes. Some hints are provided in the comments. Note that the choices you make here will impact other part of the implementation.

Decoding the Objects (3 pts). With the implementation of the heads, the stage is now set to complete the FCOS inference. You will need to further implement the *inference* function in the class of *FCOS*. A high-level sketch of this implementation is described in the comments.

Testing your Inference Code. To test the implementation of model inference, we have included a pre-trained FCOS model (using ResNet18 as the backbone) and its config file in the assignment under ./pretrained. You can test your code using this pretrained model by running
python ./eval.py ../pretrained/voc_res18.yaml ../pretrained/voc_res18.pth.tar

The inference should run for a few minutes with the expected mAP@0.5 of 60.6%. If you could not match this inference speed or mAP score, your implementation will probably need some further scrutiny. **Please include the mAP scores, inference time, and sample detection results in the writeup.**

4.3 Model Training

Our final step is to implement the training of FCOS.

Training with the Multitask Loss Function (3 pts). The key component of training lies in computing the multitask loss. This loss combines a classification loss, a regression loss, and a center-ness loss. Computing this loss involves the assignment of targets (e.g., positive/negative labels, offsets), given the feature maps and the ground truth object annotations. While our helper code will handle rest part of the training, you will need to implement the computation of the multitask loss by filling in the *compute_loss* function within the class of *FCOS*. This implementation can be a bit tricky. Again, comments in our helper code provide some hints.

An important aspect of your implementation is its efficiency. A "sloppy" implementation will leads to major increase in training time. As a reference,

training on VOC 2007 using the default ResNet34 backbone and configuration should take around 1 hour using a single T4 GPU.

Once done, you can now train your own FCOS model, following our previous instructions. While you can get some hints about the selection of training parameters from the config file of our pretrained model and the journal paper, it will be fun to play with those hyperparameters (e.g., learning rate, data augmentation, etc.). To reduce the turnaround time, we recommend using a lightweight model and/or a reduced training schedule when experimenting with the hyperparameters. In your writeup, please include (a) training curves of your model; (b) testing mAP scores; (c) sample object detection results on the test set; and (d) the model config file and the trained model.

4.4 Bonus

Our help code is a barebone implementation of FCOS with many potential improvements. We will offer bonus points for any of the following items (2 pts each with no upper bound)

- Better model design that leads to improved performance (efficiency or accuracy).
- More aggressive data augmentations that leads to improved results.
- Systematic evaluation of hyperparameters that leads to improved results.
- Any meaningful improvement of the model with demonstrated performance boost.
- Experiment with larger scale dataset that is still feasible for this assignment, e.g., miniCOCO dataset[1].

To get full bonus points, you will need to (1) include the code in the submission; (2) describe your improvement and its rationale in the writeup; and (3) demonstrate the performance in the writeup (in terms of accuracy and/or efficiency).

5 Writeup

For this assignment, and all other assignments, you must submit a project report in PDF. Every team member should send the same copy of the report. Please clearly identify the contributions of all members. For this assignment, you will need to include (a) a response to the questions (Sec 4.1); (b) a description of your implementation for model inference and some demonstration of the results (Sec 4.2); (c) a description of your implementation for model training and some demonstration of the results (Sec 4.2); and (d) (optional) any materials related to the bonus. You can also discuss anything extra you did. Feel free to add other information you feel is relevant.

[1] `https://github.com/giddyyupp/coco-minitrain`

6 Handing in

This is very important as you will lose points if you do not follow instructions. Every time after the first that you do not follow instructions, you will lose 5%. The folder you hand in must contain the following:

- code/ - directory containing all your code for this assignment
- writeup/ - directory containing your report for this assignment.
- results/ - directory containing your results. **Please copy your final model (model_final.pth.tar in your experiment folder) and its corresponding config file in this folder.** If you plan to include multiple models for bonus points, please describe each of them in the writeup. Visuals including sample detection results, training curves should not included in the writup and not in this folder.

Do not use absolute paths in your code (e.g. /user/classes/proj1). Your code will break if you use absolute paths and you will lose points because of it. Simply use relative paths as the starter code already does. **Do not turn in the data / logs / models folder**. Hand in your project as a zip file through Canvas. You can create this zip file using *python zip_submission.py*.

References

[1] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[4] Z. Tian, C. Shen, H. Chen, and T. He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9627–9636, 2019.

[5] Z. Tian, C. Shen, H. Chen, and T. He. Fcos: A simple and strong anchor-free object detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.