BMI / CS 771 Fall 2022: Homework Assignment 3

Yin Li

Nov 2022

1 Overview

One of the central problems in computer vision is to recognize and localize object instances with an input image. This problem, also 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 FCOS on PASCAL VOC 2007 [1] using your implementation. The conference version of FCOS [4] is listed in our reading material. This assignment will follow its journal version [5], which incorporates better designs and provides more details. While the model is conceptually simple, the implementation requires some careful thoughts for efficient training and inference.

This assignment is team-based and requires cloud computing. A team can have up to 3 students. The assignment has two parts with a total of 24 points and unlimited bonus points. Details and rubric are described in Section 2 and 3.

2 Part I: Critique of the Paper

The first part of the assignment is to write a short critique of [5]. The critique must be included in your writeup and address the following questions.

- What was this paper about? What was the key idea or intuition? (a summary of the paper)
- What was the design of the experiments? How was the results analyzed? (a summary of the experiments and results)
- What is (are) the major contribution(s) and finding(s)?
- What do you consider as the strength and weakness of the method? You might want to compare FCOS to other detectors that we covered in class (or you find in the literature).
- What are your suggestions to improve this research?

- Any other comments you have about the paper.

Rubric: The critique should be at least 500 words *in your own words.* A maximum of 12 points can be granted for this part. There is no bonus point.

3 Part II: Implementation of FCOS

The second part of this assignment is to implement FCOS using PyTorch and with our helper code. FCOS considers a dense labeling problem for detecting objects in images. The model design is illustrated in Figure 1 (take from Figure 2 in [5]). Here we describe the setup of our help code, and provides detailed instructions for the implementation.

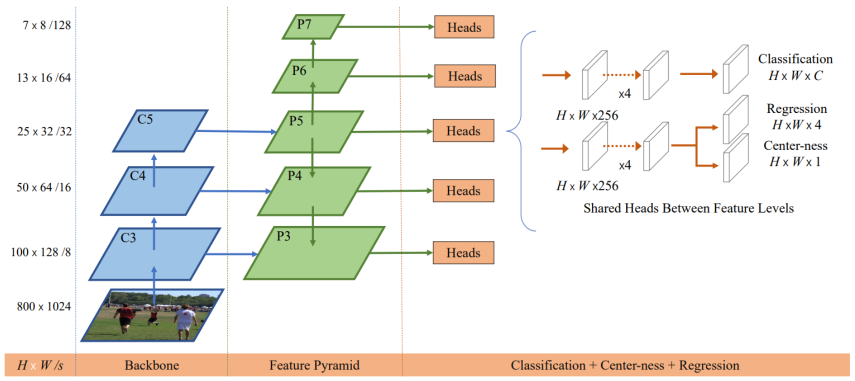

Figure 1: Overview of FCOS from [5]. The model consists of a backbone network, a feature pyramid, and shared classification / regression heads.

3.1 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, 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, 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 reference any 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. 5) and a writeup. The submission can be generated using:
python ./zip_submission.py
- An autograder will be used to grade some parts of the assignment. Please follow the instructions closely.

3.2 Implementation Details

The implementation of FCOS will have to handle technical details beyond the model itself, similar to the implementation of many vision models. We recommend going through our helper code before you start implementing the model. Our helper code is organized into four parts.

Input Pipeline (./libs/dataset.py and ./libs/transforms.py). Unlike image classification, the labels of object detection (e.g., boxes) depend on the location of the pixels. Hence, any data augmentation that changes the pixel location will have to modify the detection labels accordingly, resulting in constrains in data augmentation and complications in data batching.

For example, horizontal flipping of an input image (as a common data augmentation) will require the flipping of all boxes. Further, image cropping is less favourable, as cropping may fundamentally change the concept of an object instance (e.g., a region of a car vs. a cropped region of a car). Without using cropping and given that images can have different aspect ratios, batching multiple input images into a tensor becomes more challenging. This class also handles box resizing and rescaling.

Our helper code has implemented a dataset class (*VOCDetection*), where simple augmentations (such as horizontal flipping) can be applied to both input image and their labels. The helper code also included a *GeneralizedRCNNTransform* class that applies scale jittering, and then batches the resized images by padding them in a common shape.

Modeling (./libs/model.py). This is the implementation of the FCOS model. The model, as shown in Figure 1, consists of (a) a ResNet-18 backbone to extract image features [2]; (b) a feature pyramid network [3] to refine image features into a pyramid; and (c) shared classification and regression heads that are applied to each pyramid level and predicts object labels and boxes, all implemented in the *FCOS* class. Note that our implementation is a simplified from [5]. For example, we used a shallower backbone (ResNet-18) and did not follow the exact design of the feature pyramid. This file currently misses several critical pieces in order to function properly. You will need to complete the code here.

Training / Inference Pipeline (./libs/engine.py, ./train.py, ./eval.py). These files contain helper code for training and evaluating the model. Specifically, *engine.py* includes functions to train the model for one epoch, and to evaluate a trained-model on a test set, while *train.py* and *eval.py* provide a high-level wrapper for training and inference, respectively.

Other Utility Functions (all other files). Other helper code includes utility functions that are necessary for the project, including the implementation of the loss functions, the code for saving checkpoints, setting up the optimizer, etc. A key component is centralized parameter configuration, as defined in *./libs/config.py*, where most of the parameters of the model architecture, training, and inference are typed and can be overwritten using a external configuration file (in yaml format). We have included a basic configuration file for VOC dataset under ./configs/voc_fcos.yaml.

Your Implementation. To complete the project, you will need to fill in the missing code in *./libs/model.py*. Specifically, your will need to implement

- The forward functions of the FCOSClassificationHead and FCOSRegressionHead classes (**4 pts**). These classes implement the classification and regression heads of the FCOS detector.
- The compute_loss function of the FCOS class (**4 pts**). This function computes the desired targets of the model outputs, and calculates the loss used during training.
- The inference function of the FCOS class (**4 pts**). This function post-processes the outputs of the FOCS model, and decodes object detection results (bounding boxes, scores, and labels).

Please refer to the instructions in the code for more implementation details. An important aspect of your implementation is efficiency. As a "sloppy" implementation will leads to major inefficiency. As a reference, the training on VOC 2007 should take no more than a few hours using a single T4 GPU.

Training and Inference. Our helper code provides the full training and inference schemes.

- 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 by using
  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 as *eval_results.json* under the experiment folder. You can also specify a checkpoint by using "-epoch". See the code for more details.

Bonus. Our help code offers 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)

- Visualization of the detection results (e.g., using the result json file).
- Better model design that leads to improved performance.
- More aggressive data augmentations that leads to improved results.
- Systematic evaluation of hyperparameters that leads to improved results.
- Any meaningful improvement of the model.

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

Rubric. A total of 12 points can be granted for this part, with no limit on bounus points.

- The forward functions of the FCOSClassificationHead and FCOSRegressionHead classes (**4 pts**). The outputs will be compared against a reference implementation using the same set of weights. Partial credit is assigned if the implementation follows a meaningful logic.

- The compute_loss function of the FCOS class (**4 pts**). The mAP of a trained model will be compared against a trained model using a reference implementation. Full credit is assigned if the mAP falls into a relative 10% range of the reference model. Partial credit is assigned if the implementation follows a meaningful logic.

- The inference function of the FCOS class (**4 pts**). The runtime of a trained model will be compared against a trained model using a reference implementation. Full credit is assigned if the runtime falls into a relative 50% range of the reference model using the same hardware. Partial credit is assigned if the implementation follows a meaningful logic.

4 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. For teams with more than one member, **please clearly identify the contributions of all members**. For this assignment, you will need to include (a) a critique of the paper; and (b) a brief description of your implementation and some demonstration of the results. You can also discuss anything extra you did. Feel free to add other information you feel is relevant.

5 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) in this folder.** If you plan to include multiple models for bonus points, please describe each of them in the writeup. Additional results, including visualization of the detection outputs can be included.
- model/ - directory containing your final model.

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.