University of Michigan
EECS 442/504: Computer Vision
Fall 2022. Instructor: Andrew Owens.

Problem Set 7: Object Detection

Posted: Wednesday, October 26, 2022 **Due:** Wednesday, November 09, 2022

Please convert your Colab notebook to a PDF file and submit the PDF file to Gradescope and Canvas. We have included the PDF conversion script at the end of the notebook.

This problem set has 2 notebooks:

Part 1 for both EECS 442 & 504 students

Part 2 for EECS 504 students only

The **Part 1** starter code for EECS 442/504 students can be found at: `https://colab.research.google.com/drive/1srKuntASTULfZlz7N7KVoruUyALdMtC-?usp=sharing`

The **Part 2** starter code for only EECS 504 students can be found at: `https://drive.google.com/file/d/1Gv7ZOSv2JLqSxH983THejf3JzIXPoKKs/view?usp=sharing`

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Please note: We have provided you with 2 weeks for this problem set. While the amount of code that you will write is similar to previous problem sets, there are two reasons you may want to consider starting early. First, there is significantly more code to read and understand than usual. Second, the final model will take a longer-than-usual amount of time to train (up to 40 minutes). For this reason, it will be more important than usual to start early to avoid Colab usage limits.

Problem 7.1 *Object Detection*

We will implement a simple object detector based on FCOS [1], a single-stage object detector that uses a feature pyramid network (FPN) to produce bounding box predictions at mutiple spatial scales. Unlike in R-CNN-style models, single-stage detectors predict bounding boxes and classes without explicitly cropping region proposals out of the image or feature map, making them faster to run.

We have given you most of the code, but have left a few key functions unimplemented. Your task is to: 1) understand the model and code that we have provided, and 2) fill in these missing pieces. Consequently, this problem set will require you to read significantly more code than previous problem sets. However, the amount of code you actually write will be comparable to prior problem sets.

We'll train and evaluate our detector on a subset of the PASCAL VOC dataset, a standard dataset for object detection tasks. The full dataset contains a total of 11K train/validation data images with 27K labeled objects, spanning 20 classes (Figure 2). We will use a roughly 1/4th subset of this dataset.

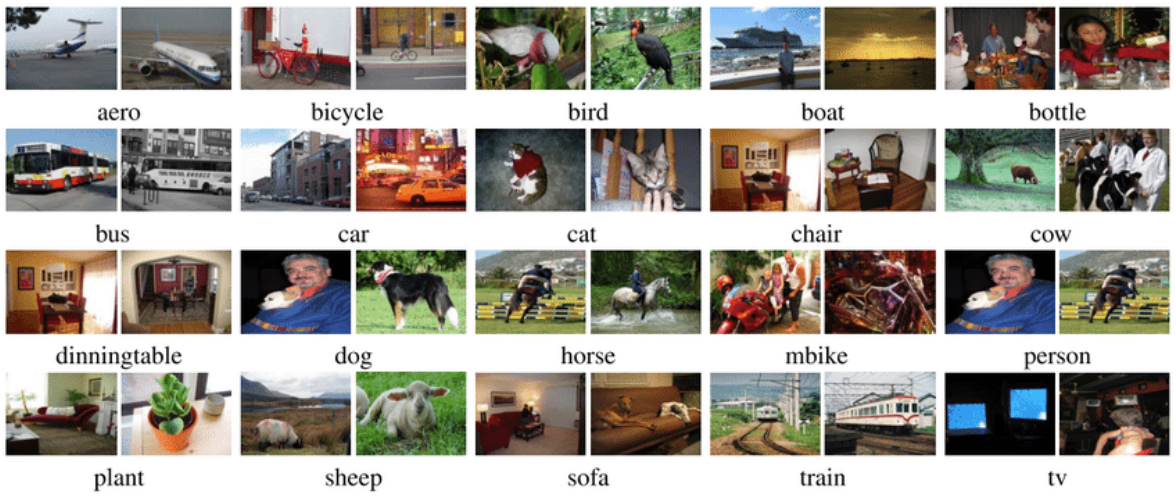

Figure 1: Example images and detections from the PASCAL VOC dataset.

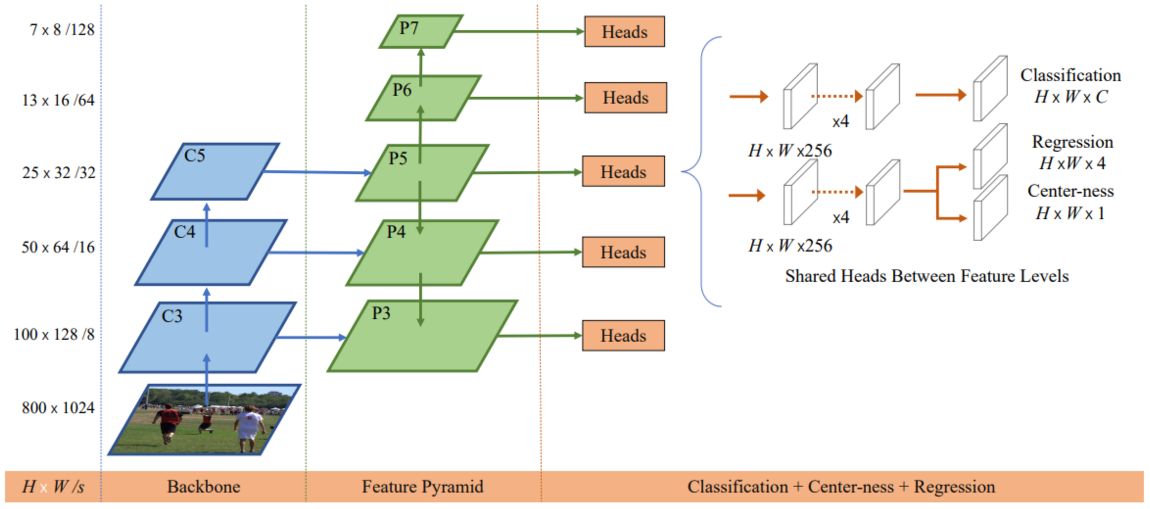

Figure 2: The FCOS architecture that we will be implementing[2]

Below, we outline the steps in the object detection pipeline and the modules that you will be implementing. Please refer to the comments in the provided notebook for further implementation details. Also, we encourage you to not necessarily jump directly to the part of the notebook that requires you to write code — instead, we recommend first reading the comments in the starter code we've provided, and to understand the utility of each function by understanding their inputs and outputs.

(a) We have provided the code for the data loader and for data preprocessing it in a separate Python file named `ps7_helpers.py`. This file will be downloaded and stored in your Colab instance and functions from it will be imported. Make sure to run the "Getting started" section every time you reconnect to your Colab instance. You can open this file and view the code for more information. Running the cells in this section will also setup the dataloader for you and show you some sample data. **(0 points)**

(b) First, we will implement the backbone and the FPN of the FCOS model. We will use RegNet-X-400MF [3] as our backbone network for image feature extraction. This is a simple convolutional network intended for efficient computation. For faster training, we will use weights (from PyTorch) pretrained on the ImageNet dataset. We will obtain 3 encoder layers, named `C3`, `C4`, and `C5` from the last layer of each of the last 3 spatial blocks of a ResNet-style network. **(2 points)**

(c) Next, we will implement the prediction head. This head has shared weights across all scales of the feature pyramid network. Its purpose is to predict an object class, bounding box, and centerness at every location. Follow the instructions in the notebook to implement the `init` and `forward` functions. **(2 points)**

(d) The FCOS heads make three predictions at every location: object class, bounding box and centerness. We need to assign GT targets for every model prediction. We want to assign a ground truth box to every FPN feature map location. Implement the function `get_fpn_location_coords` to map each point in the feature map at every FPN level back to a point in the input image. More information on this mapping can be found in the Discussion 9 slides. **(1 point)**

(e) Next, we have to assign a GT to each of these location coordinates we obtained from the above function. We have implemented this code for you in `match_locations_to_gt` function available in the `ps7_helpers.py` file. **(0 points)**

(f) The box regression head is tasked with predicting four values which are the distances from feature locations to box edges. FCOS normalizes absolute targets by the stride of FPN levels. Next, you will implement this transformation logic and its inverse in two separate functions: `fcos_get_deltas_from_locations` and `fcos_apply_deltas_to_locations` **(2 points)**

(g) We have provided you with an estimate of *centerness*, a measure of how centrally located a given position in the feature pyramid network is relative to the bounding box. This is used to determine which bounding boxes are high quality, and which are more likely to contain errors. You will not need to implement this part of the system, but may find it interesting to read **(0 points)**.

(h) FCOS has 3 prediction layers that use 3 different losses, namely: a sigmoid *focal loss*, generalized intersection-over-union (GIoU) and binary cross entropy (BCE). We will use PyTorch implementations of each loss, so you will not need to implement. To help you understand how they work, we have provided a demonstrations. **(0 points)**

(i) Finally, you'll bring everything that you have implemented together in the `FCOS` class. Follow the instructions in the notebook to implement the `init` and `forward` functions. You can leave the `inference` function blank, run the following training cells and come back to implement it after training is over. **(3 point)**

(j) To make sure that everything is working as expected, we can try to overfit the detector to a small subset of data. This overfitting experiment should only take about a few minutes to run. After 500 epochs (i.e., complete passes through the data) of training you should see a loss curve that (while not necessarily strictly decreasing) has some downward movement. Once you are confident that the training code is working properly, you'll train the full network on more data and for longer. We will train for 9000 iterations; this ideally takes about 35 minutes on the GPUs that Colab provides. You should see a total loss of around (or less than) 0.8. **(0 points)**

(k) It is time to test our detector! Go back to the `FCOS` class and ensure you have implemented the `inference` method. We have implemented a helper function that uses your `inference` method. Run the cell, and it will display the output as predicted bounding boxes on the image along with expected ground truth. **(0 points)**

(l) Lastly, it is time to measure the performance of our detector using the mean average precision (mAP) metric. This evaluation metric computes the average precision for each class, and then averages over all classes (please see the lecture note for more details). We will use an existing mAP implementation (504 students will reimplement it in a subsequent section). Run the cells for mAP and check the output. You should get around 40% mAP for full points in (i).

Submission: Follow the instructions and run the pdf conversion cell to obtain the PDF, and upload it to Gradescope. Please do not forget to upload the Colab notebook to Canvas as well.

EECS 504 only

Problem 7.2 *Inference and Evaluation Metrics for Object Detection* **(EECS 504 only)**

In this problem, we will implement the inference components and evaluation metrics needed for object detection.

1. Your first task is to implement the `IoU` function, which estimates the intersection-over-union (IoU) of two bounding boxes (also called the *Jaccard similarity*). Recall that IoU for two boxes B_1 and B_2 is:

$$\text{IoU}(B_1, B_2) = \frac{|B_1 \cap B_2|}{|B_1 \cup B_2|}, \tag{1}$$

   where $\cap$ and $\cup$ are set intersection and union operations, respectively. This function is used to calculate the IoU between the predicted and ground-truth bounding boxes. It wil be used as part of your implementation of the mAP function, and used in the Generalized IoU loss function in the object detection model. **(1 point)**

2. At test time, we will discard redundant bounding box predictions as a post processing step. Specifically, we will use *non-maximum suppression* (NMS) to remove any bounding box prediction that significantly overlaps another bounding box prediction that has a higher confidence score. Slides 47 and 48 in the object detection lecture and Discussion 9 slides describe the NMS process.(**2 points**)

3. To evaluate the performance of our object detector, we will use the mean Average Precision (mAP) metric. mAP is computed by taking the average of precision over all the

classes in the dataset. Slides 44, 45 and 46 in the object detection lecture and Discussion 9 slides gives detailed information on the process to calculate mAP.(**2 points**)

Submission: Follow the instructions and run the PDF conversion cell to obtain the PDF, and upload it to Gradescope. Please do not forget to upload the Colab notebook to Canvas as well.

Resources

1. Lecture slides
2. FCOS paper [1]
3. Lilian Weng's object detection tutorial
4. Understanding mAP

Acknowledgement

Much of this homework and the starter code was adapted from the winter 2022 version of EECS 598 Deep Learning for Computer Vision course offered by Justin Johnson.

References

[1] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: fully convolutional one-stage object detection. *CoRR*, abs/1904.01355, 2019.

[2] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: A simple and strong anchor-free object detector, 2020.

[3] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. *CoRR*, abs/2003.13678, 2020.