BME646 and ECE60146: Homework 6

Spring 2023
Due Date: 11:59pm, Mar 22, 2023
TA: Fangda Li (li1208@purdue.edu)

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace. Late submissions will be accepted with penalty: **-10 points per-late-day, up to 5 days**. **This can be a VERY challenging homework. Start early!**

1 Introduction

Why detect only one pizza when there are multiple? Unironically again, the goal of this homework is for you create your own *multi-pizza detector* that is based on the YOLO framework [2]. Your learning objectives for this HW are:

1. Understand the YOLO logic – how multi-instance object detection can be done via just a single forward pass of the network.

2. Implement your own YOLO training and evaluation logic. You might just be surprised how much more complicated the logic becomes when dealing with multi-instance rather than single-instance.

The following steps will prepare you to work with object detection, data loading with annotations, *e.g.* , bounding boxes and labels, and so on. Before starting to code, it is highly recommended that you read through the entire handout first.

2 Getting Ready for This Homework

Before embarking on this homework, do the following:

1. Your first step would be to come to terms with the basic concepts of YOLO: Compared to everything you have done so far in our DL class, the YOLO logic is very complex. As was explained in class, it uses the notion of Anchor Boxes. You divide an image into a grid of cells and you associate N anchor boxes with each cell in the grid. Each anchor box represents a bounding box with a different aspect ratio.

Your first question is likely to be: Why divide the image into a grid of cells? To respond, the job of estimating the exact location of an object is assigned to that cell in the grid whose center is closest to the center of the object itself. Therefore, in order to localize the object, all that needs to be done is to estimate the offset between the center of the cell and the center of true bounding box for the object.

But why have multiple anchor boxes at each cell of the grid? As previously mentioned, anchor boxes are characterized by different aspect ratios. That is, they are candidate bounding boxes with different height-to-width ratios. In Prof. Kak's implementation in the `RegionProposalGenerator` module, he creates five different anchor boxes for each cell in the grid, these being for the aspect ratios: `[1/5, 1/3, 1/1, 3/1, 5/1]`. The idea here is that the anchor box whose aspect ratio is closest to that of the true bounding box for the object will speak with the greatest confidence for that object.

2. You can deepen your understanding of the YOLO logic by looking at the implementation of image gridding and anchor boxes in Version 2.0.8 of Prof. Kak's `RegionProposalGenerator` module:

   https://engineering.purdue.edu/kak/distRPG/

   Go to the Example directory and execute the script:

   ```
   multi_instance_object_detection.py
   ```

   and work your way backwards into the module code to see how it works. In particular, you should pay attention to how the notion of anchor boxes is implemented in the function:

   ```
   run_code_for_training_multi_instance_detection()
   ```

   To execute the script `multi_instance_object_detection.py`, you will need to download and install the following datasets:

   ```
   Purdue_Dr_Eval_Multi_Dataset-clutter-10-noise-20-size-10000-train.gz
   Purdue_Dr_Eval_Multi_Dataset-clutter-10-noise-20-size-1000-test.gz
   ```

   Links for downloading the datasets can be found on the module's webpage. In the dataset names, a string like `size-10000` indicates the number of images in the dataset, the string `noise-20` means 20%

added random noise, and the string `clutter-10` means a maximum of 10 background clutter objects in each image.

Follow the instructions on the main webpage for `RegionProposalGenerator` on how to unpack the image data archive that comes with the module and where to place it in your directory structure. These instructions will ask you to download the main dataset archive and store it in the `Examples` directory of the distribution.

3 Programming Tasks

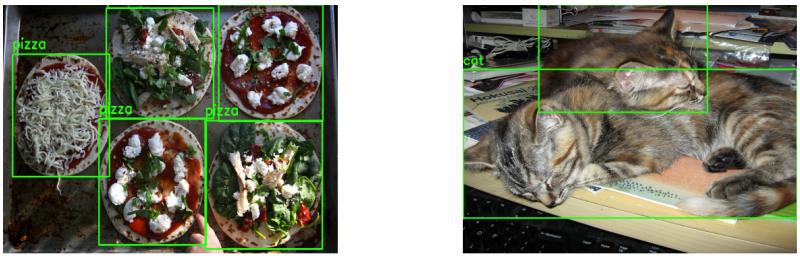
3.1 Creating Your Own Multi-Instance Object Localization Dataset

In this exercise, you will create your own dataset based on the following steps:

1. Similar to what you have done in the previous HWs, first make sure the COCO API is properly installed in your `conda` environment. As for the image files and their annotations, we will be using both the **2014 Train images** and **2014 Val images**, as well as their accompanying annotation files: **2014 Train/Val annotations**. For instructions on how to access them, you can refer back to the HW4 handout.

2. Now, your main task is to use those files to create your own multi-instance object localization dataset. More specifically, you need to write a script that filters through the images and annotations to generate your training and testing dataset such that any image in your dataset meets the following criteria:

   - Contains at least one *foreground object*. A foreground object must be from one of the three categories: `['bus', 'cat', 'pizza']`. Additionally, the area of any foreground object must be larger than $64 \times 64 = 4096$ pixels. Different from the last HW, there can be multiple foreground objects in an image since we are dealing with multi-instance object localization for this homework. If there is none, that image should be discarded. Also, note that you can use the area entry in the annotation dictionary instead of calculating it yourself.
   - When saving your images to disk, resize them to 256×256. Note that you would also need to scale the bounding box parameters accordingly after resizing.



(a) Example 1 (b) Example 2

Figure 1: Sample COCO images with bounding box and label annotations.

- Use only images from **2014 Train images** for the training set and **2014 Val images** for the testing set.

Again, you have total freedom on how you organize your dataset as long as it meets the above requirements. If done correctly, you will end up with at least 6k training images and 3k testing images.

3. In your report, make a figure of a selection of images from your created dataset. You should plot at least 3 images from each of the three classes like what is shown in Fig. 1 and with the annotations of all the present foreground objects.

3.2 Building Your Deep Neural Network

Once you have prepared the dataset, you now need to implement your deep convolutional neural network (CNN) for multi-instance object classification and localization. For this HW, you can directly base your CNN architecture on what you have implemented for HW5, which was built upon the skeleton and your own skip-connection implementation. Nonetheless, you have total freedom on what specific architecture you choose to use for this HW.

The key design choice you'll need to make is on the organization of the predicted parameters by your network. As you have learned in Prof. Kak's tutorial on Multi-Instance Object Detection [1], for any input image, your CNN should output a `yolo_tensor`. The exact shape of your predicted `yolo_tensor` is dependent on how you choose to implement image gridding and the anchor boxes. It is highly recommended that, before starting your own implementation, you should review the tutorial again and familiarize

yourself with the notions of `yolo_vector`, which is predicted for each and every anchor box, and `yolo_tensor`, which stacks all `yolo_vector`s.

In your report, designate a code block for your network architecture. Additionally, clearly state the shape of your output `yolo_tensor` and explain in detail how that shape is resulted from your design parameters, *e.g.* the total number of cells and the number of anchor boxes per cell, etc.

3.3 Training and Evaluating Your Trained Network

Now that you have finished designing your deep CNN, it is finally time to put your glorious *multi-pizza* detector in action. What is described in this section is probably the hardest part of the homework. To train and evaluate your YOLO framework, you should follow the steps below:

1. Write your own dataloader. While everyone's implementation will differ, it is recommended that the following items should be returned by your `__getitem__` method for multi-instance object localization:

   (a) The image tensor;

   (b) For each foreground object present in the image:

      i. Index of the assigned cell;

      ii. Index of the assigned anchor box;

      iii. Groundtruth `yolo_vector`.

   The tricky part here is how to best assign a cell and an anchor box given a GT bounding box. For this part, you will have to implement your own logic. Typically, one would start with finding the best cell, and subsequently, choose the anchor box with the highest IoU with the GT bounding box. You would need to pass on the indices of the chosen cell and anchor box for the calculation of the losses explained later in this section.

   It is also worthy to remind yourself that the part in a `yolo_vector` concerning the bounding box should contain four parameters: δ_x, δ_y, σ_w and σ_h. The first two, δ_x and δ_y, are simply the offsets between the GT box center and the anchor box center. While the last two, σ_w and σ_h, can be the "ratios" between the GT and anchor widths and heights:

$$w_{\text{GT}} = e^{\sigma_w} \cdot w_{\text{anchor}},$$

$$h_{\text{GT}} = e^{\sigma_h} \cdot h_{\text{anchor}}.$$

2. Write your own training code. Note that this time you will need three losses for training your network: a binary cross-entropy loss for objectness, a cross-entropy loss for classification and another loss for bounding box regression.

   The main challenge for you in this year's version of the HW is to *write your logic for loss calculation in a way that accommodates an arbitrary training batch size (i.e. greater than 1) without looping through individual images in the batch.* To walk you through the process, here is a brief summary of the steps:

   (a) For each anchor box that has been assigned a GT object, set the corresponding target objectness to one and everywhere else as zero.

   (b) Calculate the bounding box regression and class prediction losses only based on the predicted `yolo_vectors` for assigned anchor boxes. The predicted `yolo_vectors` for anchor boxes where no GT object has been assigned are simply ignored.

   (c) Note that all of the aforementioned logic can be done simultaneously and efficiently for all images in a batch just with some clever indexing and without using `for` loops.

3. Write your own evaluation code. Different than evaluating single-instance detectors, quantitatively examining the performance of a multi-instance detector can be much more complicated and may be beyond the scope of this HW, as you have learned in Prof. Kak's tutorial [1]. Therefore, for this HW, we only ask you to present your multi-instance detection and localization results qualitatively. That is, for a given test image, you should plot the predicted bounding boxes and class labels along with the GT annotations.

   More specifically, you will need to implement your own logic that converts the predicted `yolo_tensor` to bounding box and class label predictions that can be visualized. Note that for this part, you are allowed to implement your logic which may only accommodate a batch size of one.

4. In your report, write several paragraphs summarizing on how you have implemented your dataloading, training and evaluation logic. Specifically you should comment on how you have accommodated batch size greater than 1 in training. Additionally, include a plot of all three

losses over training iterations (you should train your network for at least 10 epochs).

For presenting the outputs of your YOLO detector, display your multi-instance localization and detection results for at least 8 different images from the *test set*. Again, for a given test image, you should plot the predicted bounding boxes and class labels along with the GT annotations for all foreground objects. You should strive to present your best *multi-instance* results in at least 6 images while you can use the other 2 images to illustrate the current shortcomings of your multi-instance detector. Additionally, you should include a paragraph that discusses the performance of your YOLO detector.

4 Submission Instructions

Include a typed report explaining how did you solve the given programming tasks.

1. Your pdf must include a description of
   - The figures and descriptions as mentioned in Sec. 3.
   - Your source code. Make sure that your source code files are adequately commented and cleaned up.
2. Turn in a zipped file, it should include (a) a typed self-contained pdf report with source code and results and (b) source code files (only .py files are accepted). Rename your .zip file as hw6_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too.
3. **Do NOT submit your network weights.**
4. For all homeworks, you are encouraged to use `.ipynb` for development and the report. If you use `.ipynb`, please convert it to `.py` and submit that as source code.
5. You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission. **If you are submitting late, do it only once on BrightSpace.** Otherwise, we cannot guarantee that your latest submission will be pulled for grading and will not accept related regrade requests.

6. The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**

7. To help better provide feedbacks to you, make sure to **number your figures**.

References

[1] Multi-Instance Object Detection – Anchor Boxes and Region Proposals. URL https://engineering.purdue.edu/DeepLearn/pdf-kak/MultiDetection.pdf.

[2] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.