BME646 and ECE60146: Homework 7

Spring 2025
Due Date: 11:59pm, March 12, 2025
TA: Akshita Kamsali (akamsali@purdue.edu)

Turn in typed solutions via Gradescope. Post questions to Piazza. Additional instructions can be found at the end. Late submissions will be accepted with penalty: -10 points per-late-day, up to 5 days.

This is a VERY challenging homework. Start early!

1 Introduction

This homework has two goals: (1) To show how a complex application of deep learning may require multiple loss functions, each possibly dealing with a different portion of the network output; and (2) To introduce you to the fundamental notions of the YOLO approach to multi-instance object detection. You need multi-instance object detection when an image contains multiple objects of interest and you want to predict their class labels and, at the same time, the coordinates of their bounding boxes.

2 Getting Ready for This Homework

The three following subsections are designed so that you'll start gently and then graduate on to more complex cases of object detection and localization.

2.1 Starting with Simple Single-Instance Detection

In order to get ready for this homework, you need to first familiarize yourself with the mechanics of **single-instance object detection** where we assume that there exists only one instance of an object of interest in an image. Anything else in the image is considered to be structured and/or random

noise. For a gentle-introduction to single-instance object detection, play with the following script in the Examples directory of DLStudio:

```
object_detection_and_localization.py
```

This script contains the following statement:

```
model = detector.LOADnet2(skip_connections=True, depth=8)
```

Now work your way backwards into DLStudio and familiarize yourself with DLStudio's inner class `DetectAndLocalize` that has the definition for the network `Loadnet2` called above. You will also notice that the script `object_detection_and_localization.py` uses the `PurdueShapes5` dataset of 32×32 images. This dataset is a part of what you will download if you click on the link "Download the image datasets for the main DLStudio module" on the main page for DLStudio. Slides 51-56 of the Week 7 lecture show you a sampling of the images and how they packed in the archive.

2.2 Graduating to More Complex Single-Instance Detection

After the script in Section 2.1 above has made you comfortable with simultaneously using two loss functions, `nn.CrossEntropy` for classification and `nn.MSE` for BB regression, you will be ready to experiment with more complex detection and localization scenarios with the help of your instructor's `YOLOLogic` module.

To that end, download and install the YOLOLogic module from the link:

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

and execute the following script

```
single_instance_object_detection.py
```

The dataset that is used in the script named above is described in Slides 77 through 80 of the Week 7 lecture. Each image in the dataset has one of

following three oriented objects: "Dr. Eval", "house", and "watertower". In addition, every image has significant structured clutter and noise that compete with the oriented objects that you need to detect and localize. The name of the dataset archive is

```
datasets_for_YOLO.tar.gz
```

Uncompressing this archive will deposit the following training datasets in your computer:

```
Purdue_Dr_Eval_Dataset-clutter-10-noise-20-size-10000-train.gz

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

The script named above, `single_instance_object_detection.py` uses the first of these two.

2.3 Getting Ready for Multi-Instance Object Detection with YOLO

Your fastest entry into YOLO logic would be through the following script

```
multi_instance_object_detection.py
```

This will require the second of the two datasets mentioned previously. The name of this dataset is `PurdueDrEvalMultiDataset`. Note the subsring "Multi" in the name of the daaset. Slides 111-117 of the Week 7 lecture show you sample images in this dataset. Compared to the dataset you used for the script in Section 2.2, now each image has multiple instances of the objects of interest (OOI) and your job is extract them along with their bounding boxes. The OOIs continue to be "Dr. Eval", "house" and "watertower".

The script named above invokes the code in YOLOLogic's inner class

`YoloObjectDetector`. Get as good an understanding of this class as you can since that will help you directly with this homework.

With regard to the logic of YOLO, as you know from your instructor's Week 7 slides, it is based on the notion of Image Cells and 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 your instructor's implementation in the YOLOLogic 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.

The suggestions made in Sections 2.1, 2.2, and 2.3 above should get you ready for the homework. You might wish to test your code on the rather simple datasets mentioned above before testing the code on the COCO dataset as required by Section 3.2 of this homework.

3 Programming Tasks

This section contains guidelines on how to extract images with one or more than one instance of the object from the COCO dataset. Finally, implement the YOLO logic to perform multi-instance detection.

3.1 How to Use the COCO Annotations

For this homework, you will need bounding boxes in addition to the labels from the COCO dataset.

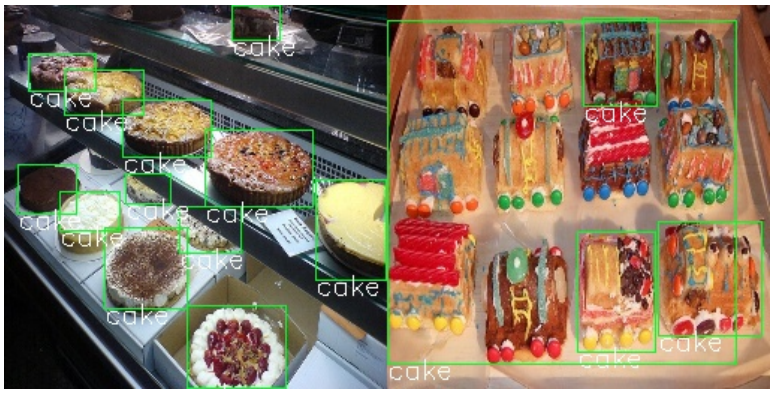

Figure 1: Sample COCO images with bounding box and label annotations for multi-instances.

In this section, we go over how to access these annotations as shown in Fig. 1. The code below is sufficient to introduce you how to prepare your own dataset and write your dataloader for this homework.

Before we jump into the code, it is important to understand structures of the COCO annotations. The COCO annotations are stored in the list of dictionaries and what follows is an example of such a dictionary:

```
{
    "id": 1409619,                  # annotation ID
    "category_id": 1,               # COCO category ID
    "iscrowd": 0,                   # specifies whether the
                                    segmentation is for a single
                                    object or for a group/cluster
                                    of objects
    "segmentation": [
        [86.0, 238.8, ..., 382.74, 241.17]
    ],                              # a list of polygon vertices
                                    around the object (x, y pixel
                                    positions)
    "image_id": 245915,             # integer ID for COCO image
    "area": 3556.2197000000015,     # Area measured in pixels
    "bbox": [86, 65, 220, 334]      # bounding box [top left x
                                    position, top left y position,
```

```
                                    width, height]
}
```

The following code (refer to inline code comments for details) shows how to access the required COCO annotation entries and display a randomly chosen image with desired annotations for visual verification. After importing the required python modules (*e.g.* `cv2`, `skimage`, `pycocotools`, etc.), you can run the given code and visually verify the output yourself.

```
# Input
input_json = 'instances_train2014.json'
class_list = ['pizza', 'cat', 'bus']

##########################
# Mapping from COCO label to Class indices
coco_labels_inverse = {}
coco = COCO(input_json)
catIds = coco.getCatIds(catNms=class_list)
categories = coco.loadCats(catIds)
categories.sort(key=lambda x: x['id'])
print(categories)


for idx, in_class in enumerate(class_list):
    for c in categories:
        if c['name'] == in_class:
            coco_labels_inverse[c['id']] = idx
print(coco_labels_inverse)

############################
# Retrieve Image list
imgIds = coco.getImgIds(catIds=catIds)

############################
# Display one random image with annotation
idx = np.random.randint(0, len(imgIds))
img = coco.loadImgs(imgIds[idx])[0]
I = io.imread(img['coco_url'])
# change from grayscale to color
if len(I.shape) == 2:
    I = skimage.color.gray2rgb(I)
# pay attention to the flag, iscrowd being set to False
annIds = coco.getAnnIds(imgIds=img['id'], catIds=catIds,
                                iscrowd=False)
anns = coco.loadAnns(annIds)
fig, ax = plt.subplots(1, 1)
image = np.uint8(I)
```

```
for ann in anns:
    [x, y, w, h] = ann['bbox']
    label = coco_labels_inverse[ann['category_id']]
    image = cv2.rectangle(image, (int(x), int(y)), (int(x + w),
                                  int(y + h)), (36, 255, 12), 2)
    image = cv2.putText(image, class_list[label], (int(x), int(
                                  y - 10)), cv2.
                                  FONT_HERSHEY_SIMPLEX,
                           0.8, (36, 255, 12), 2)
ax.imshow(image)
ax.set_axis_off()
plt.axis('tight')
plt.show()
```

3.2 Creating Your Own Multi-Instance Object Localization Dataset

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

1. You need to write a script similar to previous homeworks 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: `['pizza', 'cat', 'bus']`. Additionally, the area of any foreground object must be larger than $200 \times 200 = 40000$ pixels[1]. 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.

   - 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.

   - Again, use images from **2014 Train images** for the training set and **2014 Val images** for the testing set.

[1] Also, you can use the area entry in the annotation dictionary instead of calculating it yourself.

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 approximately 4k training images and 2k testing images

2. 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 with the annotations of all the present foreground objects. **A 3x3 grid of images.**

3.3 Building Your Deep Neural Network

1. Once you have prepared the dataset, you now need to implement your deep convolutional neural network (CNN) for multi-instance object classification and localization. You can directly base your CNN architecture on the `NetForYOLO` network in the YOLOLogic module after adjusting for YOLO parameters. Again, you have total freedom regarding your choice for the network to use. It is possible that for the COCO images, you would need a beefed up skip block for connection skipping. In addition to perhaps augmenting your instructor's `SkipBlock`, you should also look at the skip blocks in the ResNet paper:

   https://arxiv.org/abs/1512.03385

   You can access the ResNet code at:

   https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py

   As you will see, ResNet has two different kinds of skip blocks, named `BasicBlock` and `BottleNeck`. `BasicBlock` is used as a building-block in ResNet-18 and ResNet-34. The numbers 18 and 34 refer to the number of layers in these two networks. For deeper networks, ResNet uses the `BottleNeck` class.

2. 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 your instructor's Week 7 lecture, your CNN should output a `yolo_tensor` for each image at the input.

3. 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 YOLO implementation, you review the Week 7 lecture and familiarize yourself thoroughly with the notions of `yolo_vector` and `yolo_tensor`.

4. In your report, designate a code block for your network architecture.

5. 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.

3.4 Training and Evaluating Your Network

Now that you have finished designing your deep CNN, it is finally time to put your glorious *multi-object* detector in action. What is described in this section is probably the most challenging 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 *highly* 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.

Also bear in mind how exactly the BB-related parameters δ_x, δ_y, σ_w and σ_h are stored in a `yolo_vector`. 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:

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

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

2. Create your own training code (or adjust existing code) for training your network. This time, you'll need three different types of losses: a binary cross-entropy loss for detecting objects, a cross-entropy loss for classifying objects, and another loss for refining bounding box positions, MSE loss.

3. Develop your own evaluation code (or make modifications to existing code). As explained in Slides 131-136 of your instructor's Week 7 lecture, in comparison with single-instance detectors, evaluating the performance of a multi-instance detector is complex. Therefore, for this assignment, we only require you to present your multi-instance detection and localization results in a qualitative manner. This means that for each test image, you should display the predicted bounding boxes and their corresponding class labels alongside the ground truth annotations.

   Specifically, you'll need to create your own method for translating the predicted `yolo_tensor` into bounding box predictions and class label predictions that can be visually represented. You have the flexibility to implement this logic according to your own approach.

4. In your report, explain how you have implemented your dataloading, training and evaluation logic. Additionally, include a plot of all three losses over training iterations (you should train your network for at least 10-15 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 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 per class. That makes a total of atleast 18 images + 6 images = 24 images. Additionally, you should include a paragraph that discusses the performance of your YOLO detector.

4 Bonus (20pts)

Use the DIoU Loss function from DLStudio instead of the Mean Squared Error (MSE) loss for bounding box regression when training your YOLO network to reproduce the deliverables outlined in 3.4, including the training loss curves and a set of 24 annotated images. You may interact with `object_detection_and_localization_iou.py` as a sandbox to become familiar with IoU loss.

For further information on application of various IoU-based loss functions in training YOLO networks, you may refer to [1]. This paper is for your own reading.

5 Submission Instructions

Include a typed report explaining how you solved the given programming tasks. You may refer to the homework solutions posted at the class website for the previous years for examples of how to structure your report

1. **Turn in a PDF file and mark all pages on gradescope**.
2. Submit your code files(s) as zip file.
3. **Code and Output Placement:** Include the output directly next to the corresponding code block in your submission. Avoid placing the code and output in separate sections as this can make it difficult to follow.
4. **Output Requirement:** Ensure that all your code produces outputs and that these outputs are included in the submitted PDF. Submissions without outputs may not receive full credit, even if the code appears correct.

5. For this homework, you are encouraged to use `.ipynb` for development and the report. If you use `.ipynb`, please convert code to `.py` and submit that as source code. **Do NOT submit .ipynb notebooks.**

6. 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.** Otherwise, we cannot guarantee that your latest submission will be pulled for grading and will not accept related regrade requests.

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

References

[1] Xiangjie Luo, Zhihao Cai, Bo Shao, and Yingxun Wang. Unified-iou: For high-quality object detection, 2024. URL https://arxiv.org/abs/2408.06636.