BME646/ECE695DL: Homework 5

Spring 2022

Due Date: Monday, March 7, 2022 (11:59pm ET)

Extension: Monday, March 14, 2022 (25 points penalty)

Turn in your solutions via BrightSpace.

1 Introduction

This homework has the following three goals:

1. To create a CNN that carries out both classification and regression at the same time.

   As you well know already, classification means assigning a discrete label to either the entire image or to an object detected in an image.

   Regression, on the other hand, means estimating one or more numerical attributes from the image. For the purpose of this homework, the numerical attributes will be the four coordinates that define the bounding box for an object detected in the image.

2. A second major goal is for you to use a skip-block class **of your own design** for the CNN you will create for this homework.

   If you are still fuzzy about what exactly is meant by a skip block, review the Week 7 lecture notes on the subject of how to deal with vanishing gradients in deep networks.

   You will create your own skip block after familiarizing yourself with the skip blocks in the famous network ResNet and in the DLStudio module.

3. You will use COCO images and annotations for your final submission. The COCO annotations include the classification labels and the bounding boxes for various types of objects in the images.

The following steps will prepare you to work with object detection, data loading with annotations, *e.g.*, bounding boxes and labels, and so on.

2 Getting Ready for This Homework

Before embarking on this homework, do the following:

1. Review the Week 6 slides on "Using Skip Connections and ..." with the goal of understanding the relationship between the building-block class `SkipBlock` on Slides 10 through 18 and the `BMEnet` network on Slides 20 through 23. **The better you understand the relationship between the `SkipBlock` class and the `BMEnet` class in DLStudio, the faster you will zoom in on what you need to do for this homework.** Roughly speaking, you will have the same relationship between your own skip block and your network for object detection and bounding-box regression.

2. Review the Week 7 slides on "Object Detection and Localization ..." to understand how both classification and regression can be carried out simultaneously by a neural network.

   Execute the following script in the `Examples` directory of DLStudio:

   ```
   object_detection_and_localization.py
   ```

   Before you run this script, you will need to also install the following datasets that are included in the link "Download the image datasets for the main DLStudio module" at the main webpage for DLStudio:

   ```
   PurdueShapes5-10000-train.gz
   PurdueShapes5-1000-test.gz
   ```

   The integer value you see in the names of the datasets is the number of images in each. Follow the instructions on the main webpage for DLStudio on how to unpack the image data archive that comes with DLStudio 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. Subsequently, you would need to execute the following (Linux) command in the `Examples` directory:

   ```
   tar xvf datasets_for_DLStudio.tar.gz
   ```

   This will create a subdirectory `data` in the `Examples` directory and deposit all the datasets in it.

   Your own CNN for this homework should produce the sort of results that are displayed by the script `object_detection_and_localization.py`.

3. As you'll recall, the second goal of this homework asks you to conjure up a building-block class of your own design that would serve as your skip block. Towards that end, you are suppose to familiarize yourself with such classes in ResNet and in DLStudio. The better you understand the logic that goes into such building-block classes, the greater the likelihood that you'll come up with something interesting for your own skip-block class.

   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. Here is the URL to the GitHub code for ResNet:

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

3 Special Note

Since this homework asks you to conjure up your own building-block class for the skip block and also gives you freedom regarding the selection of the COCO images and at what resolution to process them, there will obviously be considerable variability in your performance numbers related to successful object detection and the accuracy of regression.

So you are very likely to wonder how we may be planning to evaluate this homework.

To forestall questions related to the above issue, note the following: You focus on your job, which is to do the homework to the best of your abilities, and we will focus on ours, which is to figure out how to best evaluate the homework submissions.

4 How to Use the COCO Annotations

For this homework, you will need labels and bounding boxes from COCO dataset. This section shows how to access and plot images with annotations as shown in Fig. 1.

The code given in this section can NOT be used as it is for completing your homework, but it should give you enough insights into COCO annotations

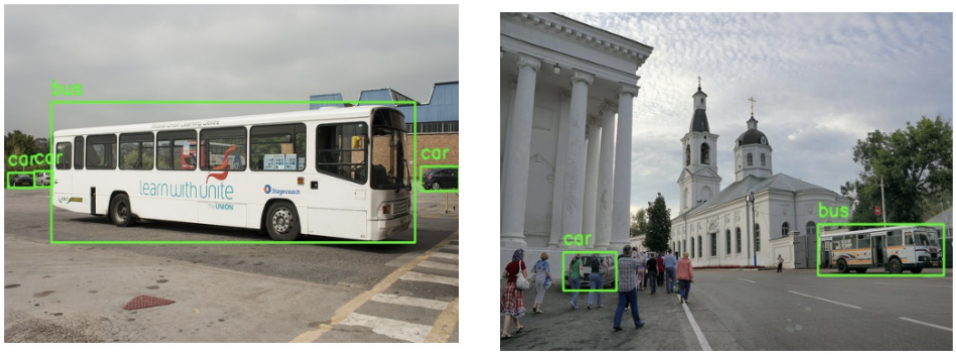
and how to access that information to write your dataloader as given in the DLStudio module.

First of all, it's important to understand some key entries in COCO annotations. The COCO annotations are stored in the list of dictionaries and each dictionary has the following key entries.

```
  annotations = [
  ...
  {
    'segmentation' : a list of polygon vertices
                  around the object (x, y pixel positions),
    'area' : Area measured in
             pixels,
    'image_id' : integer ID for COCO image,
    'bbox' : bounding box
          [top left x position, top left y position, width, height],
    'id' : annotation ID,
    'category_id' : COCO category ID,
    'iscrowd' : specifies whether the segmentation is for a single
                object or for a group/cluster  of objects,
  }
...
    ]
```

The following COCO annotation example shows multiple available entries in the form of python dictionary and the highlighted fields are of interest for this homework.

```
{'segmentation': [[234.27, 269.47, 243.97, 261.23, 277.93, 258.32,
286.66, 262.2, 287.63, 270.44, 292.48, 280.63, 289.08, 290.33, 285.6
295.67, 271.62, 295.67, 271.62, 284.03, 264.83, 274.32, 254.16,
272.38, 249.79, 279.66, 249.31, 293.73, 246.4, 298.09, 240.09,
295.18]],
'area': 1393.4401499999994, 'iscrowd': 0, 'image_id': 93611,
'bbox': [234.27, 258.32, 58.21, 39.77],'category_id': 3, 'id': 135286}
{'segmentation': [[612.0, 199.47,
539.47, 200.98, 539.47, 200.98, 477.51, 231.2, 472.98, 314.31, 483.5
326.4, 488.09, 315.82, 515.29, 321.87, 515.29, 335.47, 528.89, 344.5
533.42, 330.93, 612.0, 324.89]],
'area': 16290.817099999998,'iscrowd': 0, 'image_id': 93611,
'bbox': [472.98, 199.47, 139.02,145.06], 'category_id': 6, 'id': 1794196}
{'segmentation': [[393.06,
280.24, 419.43, 275.22, 419.43, 264.23, 409.07, 257.32, 391.49,
```



(a) Example 1 (b) Example 2

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

```
258.58, 384.58, 266.74, 380.19, 275.85, 386.78, 280.56, 392.74,
281.18]],
'area': 678.5678000000011, 'iscrowd': 0, 'image_id': 93611,
'bbox': [380.19, 257.32, 39.24, 23.86], 'category_id': 3, 'id':
2036742}
```

The following code (ref. inline code comments) 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 (ref. Fig. 1). Feel free to adjust the class list or experiment with image/annotation resizing, if you choose to resize images in your implementation.

```
#Input
input_json = 'instances_train2017.json'
class_list = ['bus','car']
############################
#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)
#[{'supercategory': 'vehicle', 'id': 3, 'name': 'car'}, {'
                                supercategory': 'vehicle', 'id
                                ': 6, 'name': 'bus'}]
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)
```

```
#{coco_cat_id:index}
#{6:0,3:1}
##############################
#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'])
if len(I.shape) == 2:
    I = skimage.color.gray2rgb(I)
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()
```

While dealing with COCO images in Homework 4, you might have come across the issue of multiple objects of different categories (e.g. cat, dog, ...) or multiple objects of the same category (e.g. cat, cat, ...) being present in the same image. In Homework 4, you could circumvent this issue by permitting the same image to be present in two or more different image folders, corresponding to their respective image labels (as long as the image contained these categories). With Homework 5, things get a bit more challenging. Here, each image has multiple image classes/categories with a variety of heights and widths. You should **only** accept an image to belong to a particular category if the height and width of its bounding box is at least 1/3 of the image's height and width. For example, if my image height is 128×128 and I am interested in downloading images of a cat. Then, the height of the bounding box denoting a cat should at least be 42. The same applies for the width. You could try using 1/2 instead of 1/3 of the image size, but some categories might not have sufficient images to populate your data. Remember, since COCO is a multi-object dataset, so if your image

in one class has larger objects of a different category, then the classifier will struggle to learn. An entirely different approach would be to only assign the largest object in an image to its category and ignore everything else. You have the freedom to employ your own approach. Use this freedom wisely.

5 Submission Instructions

You can assume that the COCO annotation files exist locally.

- Make sure to submit your code in Python 3.x and not Python 2.x.
- Create a .zip archive with the following required files:

  `hw05_training.py`

  `hw05_validation.py`

  pdf report(see the submission template)

  and optionally any additional helper python modules such as `model.py`, `dataloader.py`, etc. and upload it onto the assignment link on BrightSpace.
- **Please do NOT include your trained model in your submission:**
  `net.pth`

  We will be executing your submitted code to generate these files for verification during validation.
- **Your code must be your own work.** We will use your source code for plagiarism detection and verification of performance. Submission of both your source code and the report (in pdf) is mandatory to receive a grade.
- You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission.