BME646 and ECE60146: Homework 5

Spring 2023
Due Date: 11:59pm, Mar 06, 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 challenging homework. Start early!**

1 Introduction

The main goal of this HW is for you to create your own, unironically, **pizza detector**. To do so, you'll need to:

1. Implement your own Skip-Connection block or ResBlock. Use that block to implement a deep network for extracting convolutional features of an input image.

2. Using the deep features extracted by your deep network, implement additional layers for predicting the class label and the bounding box parameters of the dominant object in an image.

3. Incorporate the CIoU (Complete IoU) Loss in your object detection network. For this you can simply call the PyTorch class for CIoU available at

   https://pytorch.org/vision/stable/generated/torchvision.ops.complete_box_iou_loss.html

   This would be for the purpose of comparing your L_2-Loss based results with the CIoU Loss based results.

4. Implement the logic for training and evaluating your deep neural network.

Just like HW4, you will again create your own dataset based on the COCO dataset according to the guidelines specified later in this homework.

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 14 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.

3. Before you run the demo script for object detection and localization in DLStudio, 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
   ```

   Alternatively, you can also download them directly by clicking the link below:

   https://engineering.purdue.edu/kak/distDLS/datasets_for_DLStudio.tar.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.

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

```
object_detection_and_localization.py
```

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

5. 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 [2]:

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

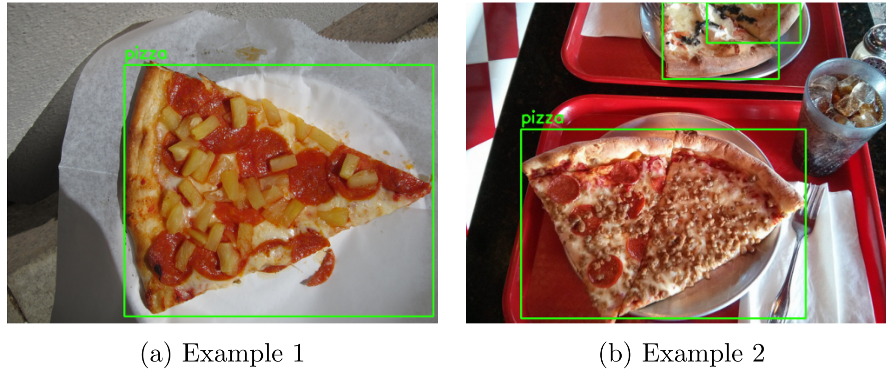
6. In this homework, you will also be comparing two different loss functions for the regression loss: The L_2-norm based loss as provided by `torcn.nn.MSELoss` and the CIoU Loss as provided by PyTorch's `complete_box_iou_loss` that is available at the link supplied in the Intro section. To prepare for this comparison, review the material on Slides 32 through 41 of the Week 7 slides on Object Detection.

3 How to Use the COCO Annotations

For this homework, you will need labels and bounding boxes from the COCO dataset. This section shows how to access and plot images with annotations as shown in Fig. 1. The code given in this section should give you enough insights into COCO annotations and how to access that information to prepare your own dataset and write your dataloader for this homework.

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 what follows is an example of such a dictionary:

```
1 {
2     "id": 1409619,              # annotation ID
```



(a) Example 1 (b) Example 2

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

```
3       "category_id": 1,             # COCO category ID
4       "iscrowd": 0,                 # specifies whether the
                                      segmentation is for a single
                                      object or for a group/cluster
                                      of objects
5       "segmentation": [
6           [86.0, 238.8, ..., 382.74, 241.17]
7       ],                            # a list of polygon vertices
                                      around the object (x, y pixel
                                      positions)
8       "image_id": 245915,           # integer ID for COCO image
9       "area": 3556.2197000000015,   # Area measured in pixels
10      "bbox": [86, 65, 220, 334]    # bounding box [top left x
                                      position, top left y position,
                                      width, height]
11  }
```

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.

```
1 # Input
2 input_json = 'instances_train2014.json'
3 class_list = ['pizza', 'bus', 'cat']
4 ###########################
5 # Mapping from COCO label to Class indices
6 coco_labels_inverse = {}
7 coco = COCO(input_json)
8 catIds = coco.getCatIds(catNms=class_list)
9 categories = coco.loadCats(catIds)
```

```python
categories.sort(key=lambda x: x['id'])
print(categories)
# [{'supercategory': 'vehicle', 'id': 6, 'name': 'bus'}, {'supercategory': 'animal', 'id': 17, 'name': 'cat'}, {'supercategory': 'food', 'id': 59, 'name': 'pizza'}]
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)
# {6: 0, 17: 1, 59: 2}
#############################
# 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()
```

4 Programming Tasks

4.1 Creating Your Own 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 HW4, 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 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 object from any of the following three categories: `['bus', 'cat', 'pizza']`.
   - Contains one *dominant object* whose bounding box area exceeds $200 \times 200 = 40000$ pixels. *The dominant object in an image is defined as the one object with the largest area and is from any of the aforementioned three classes.* Note that there can be only at most one dominant object in an image since we are dealing with single object localization for this homework. If there is none, that image should be discarded. Such images shall become useful in a future homework dealing with multi-instance localization. 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.
   - 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 roughly 4k training images and 2k 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 but only with the annotation of the dominant object.

4.2 Building Your Deep Neural Network

Once you have prepared the dataset, you now need to implement your deep convolutional neural network (CNN) for simultaneous object classification and localization. The steps for creating your CNN are as follows:

1. You must first create your own Skip-Connection Block or ResBlock. You can refer to how it is written in either DLStudio or ResNet in Torchvision [2]. **However, your implementation must be your own.**

2. The next step is to use your ResBlock to create your deep CNN. Your deep CNN should input an image and predict the following two items for the dominant object in the image:

   - The class label, similar to what you have done in HW4.
   - The bounding box parameters in the following order: `[x1, y1, x2, y2]`, where `(x1, y1)` is the location of the top left corner of the bounding box, and `(x2, y2)` is the location of the bottom right corner.

3. Again, you have total freedom on how you design your CNN for this task. Nonetheless, here is a recommended skeleton for building your network (inspired by [1]):

```
1  import torch
2  from torch import nn
3
4  class HW5Net(nn.Module):
5      """Resnet-based encoder that consists of a few
6      downsampling + several Resnet blocks as the backbone
7      and two prediction heads.
8      """
9
10     def __init__(self, input_nc, output_nc, ngf=8,
                                   n_blocks=4):
11         """
12         Parameters:
13             input_nc (int)      -- the number of channels
                                   in input images
```

```
            output_nc (int)      -- the number of channels
                                 in output images
            ngf (int)            -- the number of filters
                                 in the first conv layer
            n_blocks (int)       -- the number of ResNet
                                 blocks
        """
        assert (n_blocks >= 0)
        super(HW5Net, self).__init__()
        # The first conv layer
        model = [nn.ReflectionPad2d(3),
                 nn.Conv2d(input_nc, ngf, kernel_size=7,
                                 padding=0),
                 nn.BatchNorm2d(ngf),
                 nn.ReLU(True)]
        # Add downsampling layers
        n_downsampling = 4
        for i in range(n_downsampling):
            mult = 2 ** i
            model += [nn.Conv2d(ngf * mult, ngf * mult * 2
                                 , kernel_size=3, stride=2,
                                  padding=1),
                      nn.BatchNorm2d(ngf * mult * 2),
                      nn.ReLU(True)]
        # Add your own ResNet blocks
        mult = 2 ** n_downsampling
        for i in range(n_blocks):
            model += [ResnetBlock(...)]
        self.model = nn.Sequential(*model)
        # The classification head
        class_head = [
            ...
        ]
        self.class_head = nn.Sequential(*class_head)
        # The bounding box regression head
        bbox_head = [
            ...
        ]
        self.bbox_head = nn.Sequential(*bbox_head)

    def forward(self, input):
        ft = self.model(input)
        cls = self.class_head(ft)
        bbox = self.bbox_head(ft)
        return cls, bbox
```

4. No matter how your CNN is built, it should be “deep enough” — that is, it should contain *at least 50* learnable layers. More specifically, you

can check the number of learnable layers using the following statement:

```
num_layers = len(list(net.parameters()))
```

5. In your report, designate a code block listing your `ResBlock` and your `HW5Net` implementations. Make sure they are commented in detail. Additionally, report the total number of learnable layers in your network.

4.3 Training and Evaluating Your Trained Network

Now that you have finished designing your deep CNN, it is finally time to put your glorious pizza detector in action. To do so, you'll need the following steps:

1. Write your own dataloader similar to what you did in HW4. For single-instance object localization, your dataloader should return not only the image and its label, but also the groundtruth bounding box parameters: `[x1, y1, x2, y2]`. Note that you should make sure the coordinate values reside in the range `(0, 1)`. Additionally, if there is any geometrical augmentation taking place, the bounding box parameters would also need to be updated accordingly.

2. Write your own training code. Note that this time you will need two losses for training your network: a cross-entropy loss for classification and another loss for bounding box regression. More specifically for the latter, you'll need to experiment with two different losses: the mean squre error (MSE) loss and the Complete IoU loss. Note that if `ops.complete_box_iou` isn't available in your installed Torchvision, you can alternatively use `ops.generalized_box_iou` instead.

3. Write your own evaluation code. To quantitative evaluation of your trained pizza (and bus and cat) detector, first report the confusion matrix on the testing set for classification similar to what you have done in HW4. Subsequently, report the mean Intersection over Union (IoU) for bounding box regression (*i.e.* localization). You might find the bounding box operators in `torchvision` very useful for calculating the bounding box IoU: https://pytorch.org/vision/main/ops.html#box-operators.

4. In your report, include the confusion matrix as well as the overall classification accuracy of your pizza detector on the testing set. Additionally, report two mean IoU values of your pizza detector, trained

with the MSE-based bounding box regression loss or the CIoU-based loss. For visualization, display at least 3 images from each of the three classes with both the GT annotation (*i.e.* class label and bounding box) and the predicted annotation of the dominant object. Those images can be a mixture of successful cases as well as failed cases. Include a paragraph discussing the performance of your pizza detector and how you think can further improve it.

5 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. 4.
   - 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 hw5_<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] pytorch-CycleGAN-and-pix2pix. URL https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix.

[2] Torchvision ResNet. URL https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py.