BME 646/ ECE695DL: Homework 6

Varun Aggarwal

March 29, 2022

1 Introduction

This project was aimed at implementing YOLO network logic. In addition, training and validation procedure has to be demonstrated.

2 Methodology

This assignment was more open ended than the others. Based on the directions of the instructor, the project could have been taken into multiple directions. Based on my judgment some of the directions were:

1. Setting up dataloader
2. Setting up the network
3. Modifying loss function
4. Generalizing script for different batch size
5. Setting up evaluation matrix and image visualizations

I decided to attempt setting up the dataloader and generalizing the script for variable batch size. My motivation stemmed from the fact that training on batch size 1 was unnecessarily time consuming.

Task 1

In preparing the dataloader, script was homework 5 was modified. Essentially, when initializing the dataloader, the script checks for existing image dictionary (this behavior can be overridden with loadDict argument). If not found, it invokes COCOdownlaod function. COCOdownload function initiates image data download for valid images (images which fit the criterion) and creates a dictionary for subsequent use.

Task 2

The challenge for writing a script for generalized batch size was I didn't intend to utilize a loop for batch calculation. I wanted to use simple matrix broadcasting to reduce loop overhead. The training loss curve is given in Figure 1.

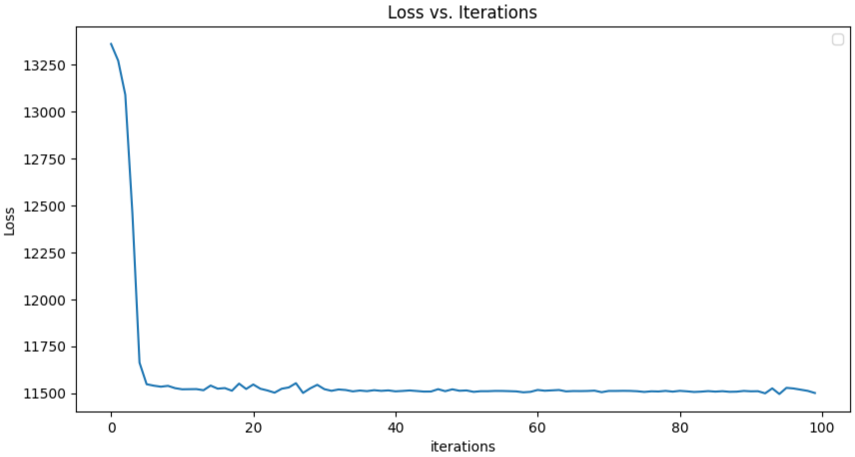

Figure 1: Skip Block for the CNN

Overall,the network was able to reduce training loss and achieved a training accuracy of 40% while a testing accuracy of 38%. The training time with batch size 1 was close to 50 mins for 10 epochs on RTX 3090 GPU while with batch size of 71, it was mere 70 s.

3 Implementation

CODE-hw06_training.py

```
from distutils.log import debug
import random, time
import numpy as np
import torch
import os, sys

from pycocotools.coco import COCO
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torch.optim as optim
import torchvision.transforms as tvt
import torchvision.transforms.functional as F
import torchvision.utils as tutils


from PIL import Image
from PIL import ImageDraw
from PIL import ImageTk
from PIL import ImageFont

import sys,os,os.path,glob,signal
import re
import functools
import math
import random
import copy
import gzip
import pickle

if sys.version_info[0] == 3:
    import tkinter as Tkinter
    from tkinter.constants import *
else:
    import Tkinter
    from Tkconstants import *
```

```
import matplotlib.pyplot as plt
import logging

# seed = 0
# random.seed(seed)
# torch.manual_seed(seed)
# torch.cuda.manual_seed(seed)
# numpy.random.seed(seed)
# torch.backends.cudnn.deterministic=True
# torch.backends.cudnn.benchmarks=False
# os.environ[’PYTHONHASHSEED’] = str(seed)

# USER imports
sys.path.append("/home/varun/work/courses/why2learn/hw/RPG-2.0.6")
from RegionProposalGenerator import *

from model import pneumaNet
from dataloader import CocoDetection
import utils


rpg = RegionProposalGenerator(
                dataroot_train = "/home/varun/work/courses/why2learn/hw/hw6
                   ↪ /data/",
                dataroot_test = "/home/varun/work/courses/why2learn/hw/hw6/
                   ↪ data/",
                image_size = [120,120],
                yolo_interval = 20,
                path_saved_yolo_model = "/home/varun/work/courses/why2learn
                   ↪ /saved_yolo_model_lr-4_bs_71_depth16.pt",
                momentum = 0.5,
                learning_rate = 1e-4,
                epochs = 50,
                batch_size = 71,
                classes = [’car’,’motorcycle’,’stop␣sign’],
                use_gpu = True,
            )

class batchedYOLO(RegionProposalGenerator.YoloLikeDetector):
    def __init__(self, rpg):
        super().__init__(rpg)
```

```
def save_yolo_model(self, acc_max, acc, model, path):
    if acc_max != 0:
        oldSaveName = "model_" + str(self.rpg.epochs) + "_" + str(self.
            ↪ rpg.learning_rate) + "_" + str(self.rpg.batch_size) + "_"
            ↪ + str(model.depth) +"_" + str(np.round(acc_max,2)) + "
            ↪ _best.pt"
        oldSaveName = os.path.join(path,oldSaveName)
        os.remove(oldSaveName)
    saveName = "model_" + str(self.rpg.epochs) + "_" + str(self.rpg.
        ↪ learning_rate) + "_" + str(self.rpg.batch_size) + "_" + str(
        ↪ model.depth) +"_" + str(np.round(acc,2)) + "_best.pt"
    saveName = os.path.join(path,saveName)
    torch.save(model, saveName)

def run_code_for_training_multi_instance_detection(self, net,
    ↪ display_labels=False, display_images=False):
    yolo_debug = False
    filename_for_out1 = "performance_numbers_" + str(self.rpg.epochs) +
        ↪ "_" + str(self.rpg.learning_rate) + "_" + str(self.rpg.
        ↪ batch_size) + "_" + str(model.depth) + "_"+"label.txt"
    filename_for_out1 = os.path.join("/home/varun/work/courses/why2learn
        ↪ /hw/hw6/runs",filename_for_out1)
    FILE1 = open(filename_for_out1, 'w')
    net = net.to(self.rpg.device)
    criterion1 = nn.BCELoss() # For the first element of the 8 element
        ↪ yolo vector ## (3)
    criterion2 = nn.MSELoss() # For the regiression elements (indexed
        ↪ 2,3,4,5) of yolo vector ## (4)
    criterion3 = nn.CrossEntropyLoss() # For the last three elements of
        ↪ the 8 element yolo vector ## (5)
    print("\n\nLearning␣Rate:␣", self.rpg.learning_rate)
    optimizer = optim.SGD(net.parameters(), lr=self.rpg.learning_rate,
        ↪ momentum=self.rpg.momentum) ## (6)
    print("\n\nStarting␣training␣loop...\n\n")
    start_time = time.perf_counter()
    Loss_tally = []
    elapsed_time = 0.0
    yolo_interval = self.rpg.yolo_interval ## (7)
    num_yolo_cells = (self.rpg.image_size[0] // yolo_interval) * (self.
        ↪ rpg.image_size[1] // yolo_interval) ## (8)
    num_anchor_boxes = 5 # (height/width) 1/5 1/3 1/1 3/1 5/1 ## (9)
```

```
max_obj_num = 5 ## (10)
## The 8 in the following is the size of the yolo_vector for each
   ↪ anchor-box in a given cell. The 8 elements
## are: [obj_present, bx, by, bh, bw, c1, c2, c3] where bx and by
   ↪ are the delta diffs between the centers
## of the yolo cell and the center of the object bounding box in
   ↪ terms of a unit for the cell width and cell
## height. bh and bw are the height and the width of object
   ↪ bounding box in terms of the cell height and width.
acc_max = 0
for epoch in range(self.rpg.epochs): ## (11)
    # print("Epoch %d"%(epoch))
    running_loss = 0.0 ## (12)
    for iter, data in enumerate(self.train_dataloader):
        if yolo_debug:
            print("\n\n\n========================================␣
               ↪ iteration:␣%d␣
               ↪ ========================================\n" % iter)
        yolo_tensor = torch.zeros( self.rpg.batch_size,
           ↪ num_yolo_cells, num_anchor_boxes, 8 ) ## (13)
        im_tensor, seg_mask_tensor, bbox_tensor, bbox_label_tensor,
           ↪ num_objects_in_image = data ## (14)
        # local_batch_size =
        im_tensor = im_tensor.to(self.rpg.device) ## (15)
        seg_mask_tensor = seg_mask_tensor.to(self.rpg.device)
        bbox_tensor = bbox_tensor.to(self.rpg.device)
        bbox_label_tensor = bbox_label_tensor.to(self.rpg.device)
        yolo_tensor = yolo_tensor.to(self.rpg.device)
        if yolo_debug:
            logger = logging.getLogger()
            old_level = logger.level
            logger.setLevel(100)
            plt.figure(figsize=[15,4])
            plt.imshow(np.transpose(torchvision.utils.make_grid(
               ↪ im_tensor,normalize=True,padding=3,pad_value=255).
               ↪ cpu(), (1,2,0)))
            plt.show()
            logger.setLevel(old_level)
        cell_height = yolo_interval ## (16)
        cell_width = yolo_interval ## (17)
        if yolo_debug:
            print("\n\nnum_objects_in_image:␣")
    print(num_objects_in_image)
num_cells_image_width = self.rpg.image_size[0] //
    ↪ yolo_interval ## (18)
num_cells_image_height = self.rpg.image_size[1] //
    ↪ yolo_interval ## (19)
height_center_bb = torch.zeros(im_tensor.shape[0], 1).float()
    ↪ .to(self.rpg.device) ## (20)
width_center_bb = torch.zeros(im_tensor.shape[0], 1).float().
    ↪ to(self.rpg.device) ## (21)
obj_bb_height = torch.zeros(im_tensor.shape[0], 1).float().to
    ↪ (self.rpg.device) ## (22)
obj_bb_width = torch.zeros(im_tensor.shape[0], 1).float().to(
    ↪ self.rpg.device) ## (23)

## idx is for object index
for idx in range(max_obj_num): ## (24)
    ## In the mask, 1 means good image instance in batch, 0
        ↪ means bad image instance in batch
    batch_mask = torch.ones( self.rpg.batch_size, dtype=torch.
        ↪ int8).to(self.rpg.device)
    if yolo_debug:
        print("\n\n␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣================␣␣object␣
            ↪ indexed␣%d␣===============␣␣␣␣␣␣␣␣␣␣␣␣␣␣\n\n" %
            ↪ idx)
    ## Note that the bounding-box coordinates are in the (x,
        ↪ y) format, with x-positive going to
    ## right and the y-positive going down. A bbox is
        ↪ specified by (x_min,y_min,x_max,y_max):
    if yolo_debug:
        print("\n\nshape␣of␣bbox_tensor:␣", bbox_tensor.shape)
        print("\n\nbbox_tensor:")
        print(bbox_tensor)
    ## in what follows, the first index (set to 0) is for
        ↪ the batch axis
    height_center_bb = (bbox_tensor[:,idx,1] + bbox_tensor[:,
        ↪ idx,3]) // 2 ## (25)
    width_center_bb = (bbox_tensor[:,idx,0] + bbox_tensor[:,
        ↪ idx,2]) // 2 ## (26)
    obj_bb_height = bbox_tensor[:,idx,3] - bbox_tensor[:,idx
        ↪ ,1] ## (27)
    obj_bb_width = bbox_tensor[:,idx,2] - bbox_tensor[:,idx,0]
        ↪  ## (28)
```

```
yolo_tensor_aug = torch.zeros(self.rpg.batch_size,
   ↪ num_yolo_cells, \
                                       num_anchor_boxes
                                          ↪ ,9).float().
                                          ↪ to(self.rpg.
                                          ↪ device) ##
                                          ↪ (55)
if (torch.any(obj_bb_height < 4.0)) or (torch.any(
   ↪ obj_bb_width < 4.0)): continue ## (29)

cell_row_indx = (height_center_bb / yolo_interval).int() #
   ↪ # for the i coordinate ## (30)
cell_col_indx = (width_center_bb / yolo_interval).int() ##
   ↪  for the j coordinates ## (31)
cell_row_indx = torch.clamp(cell_row_indx, max=
   ↪ num_cells_image_height - 1) ## (32)
cell_col_indx = torch.clamp(cell_col_indx, max=
   ↪ num_cells_image_width - 1) ## (33)

## The bh and bw elements in the yolo vector for this
   ↪ object: bh and bw are measured relative
## to the size of the grid cell to which the object is
   ↪ assigned. For example, bh is the
## height of the bounding-box divided by the actual
   ↪ height of the grid cell.
bh = obj_bb_height.float() / yolo_interval ## (34)
bw = obj_bb_width.float() / yolo_interval ## (35)

## You have to be CAREFUL about object center
   ↪ calculation since bounding-box coordinates
## are in (x,y) format --- with x-positive going to the
   ↪ right and y-positive going down.
obj_center_x = (bbox_tensor[:,idx,2].float() + bbox_tensor
   ↪ [:,idx,0].float()) / 2.0 ## (36)
obj_center_y = (bbox_tensor[:,idx,3].float() + bbox_tensor
   ↪ [:,idx,1].float()) / 2.0 ## (37)
## Now you need to switch back from (x,y) format to (i,j
   ↪ ) format:
yolocell_center_i = cell_row_indx*yolo_interval + float(
   ↪ yolo_interval) / 2.0 ## (38)
yolocell_center_j = cell_col_indx*yolo_interval + float(
   ↪ yolo_interval) / 2.0 ## (39)
```

```
del_x = (obj_center_x.float() - yolocell_center_j.float())
   ↪  / yolo_interval ## (40)
del_y = (obj_center_y.float() - yolocell_center_i.float())
   ↪  / yolo_interval ## (41)
class_label_of_object = np.array(bbox_label_tensor[:,idx].
   ↪ tolist()) ## (42))
## When batch_size is only 1, it is easy to discard an
   ↪ image that has no known objects in it.
## To generalize this notion to arbitrary batch sizes,
   ↪ you will need a batch mask to indicate
## the images in a batch that should not be considered
   ↪ in the rest of this code.

## update the batch_mask - set to zero is class is 13 i.
   ↪ e. no object present
batch_mask = batch_mask.masked_fill_(torch.BoolTensor([1
   ↪ if i==13 else 0 for i in class_label_of_object]).to
   ↪ (self.rpg.device),0)
batch_mask_index = np.where(np.array(batch_mask.tolist())
   ↪ ==1)[0]
# if class_label_of_object == 13: continue
yolo_vector = torch.zeros((self.rpg.batch_size,8), dtype=
   ↪ torch.float32)
anch_box_index = np.zeros(self.rpg.batch_size)
for ibx in range(len(batch_mask.tolist())):
    if ibx in batch_mask_index:
        AR = obj_bb_height[ibx].float() / obj_bb_width[ibx
           ↪ ].float() ## (44)
        if AR <= 0.2: anch_box_index[ibx] = 0 ## (45)
        if 0.2 < AR <= 0.5: anch_box_index[ibx] = 1 ##
           ↪ (46)
        if 0.5 < AR <= 1.5: anch_box_index[ibx] = 2 ##
           ↪ (47)
        if 1.5 < AR <= 4.0: anch_box_index[ibx] = 3 ##
           ↪ (48)
        if AR > 4.0: anch_box_index[ibx] = 4 ## (49)
# yolo_vector = torch.FloatTensor([0,del_x.item(), del_y
   ↪ .item(), bh.item(), bw.item(), 0, 0, 0] ) ## (50)
# yolo_vector[batch_mask_index,0] = torch.FloatTensor([
   ↪ self.rpg.batch_size, 0,del_x.item(), del_y.item(),
   ↪  bh.item(), bw.item(), 0, 0, 0] )
yolo_vector[batch_mask_index,0] = 1
```

```
    yolo_vector[batch_mask_index,1:5] = torch.transpose(torch.
        ↪ FloatTensor([del_x.tolist(), del_y.tolist(), bh.
        ↪ tolist(), bw.tolist()]),0,1) ## (51)
    yolo_vector[batch_mask_index,5 + class_label_of_object] =
        ↪ 1 ## (52)
    yolo_cell_index = cell_row_indx * num_cells_image_width +
        ↪ cell_col_indx ## (53)
    yolo_tensor[batch_mask_index,yolo_cell_index.tolist(),
        ↪ anch_box_index] = yolo_vector.to(self.rpg.device) #
        ↪ # (54)
    yolo_tensor_aug = torch.zeros(self.rpg.batch_size,
        ↪ num_yolo_cells, \
                                                 num_anchor_boxes
                                                     ↪ ,9).float().
                                                     ↪ to(self.rpg.
                                                     ↪ device) ##
                                                     ↪ (55)
    yolo_tensor_aug[batch_mask_index,:,:,:-1] = yolo_tensor ##
        ↪  (56)
    if yolo_debug:
        print("\n\nyolo_tensor␣specific:␣")
        print(yolo_tensor[0,18,2])
        print("\nyolo_tensor_aug_aug:␣")
        print(yolo_tensor_aug[0,18,2])
## If no object is present, throw all the prob mass into the
    ↪  extra 9th ele of yolo_vector
for ibx in range(self.rpg.batch_size):
    for icx in range(num_yolo_cells): ## (57)
        for iax in range(num_anchor_boxes): ## (58)
            if yolo_tensor_aug[ibx,icx,iax,0] == 0: ## (59)
                yolo_tensor_aug[ibx,icx,iax,-1] = 1 ## (60)
if yolo_debug:
    logger = logging.getLogger()
    old_level = logger.level
    logger.setLevel(100)
    plt.figure(figsize=[15,4])
    plt.imshow(np.transpose(torchvision.utils.make_grid(
        ↪ im_tensor, normalize=True,
                                                         padding=3,
                                                             ↪ pad_value
                                                             ↪ =255)
                                                             ↪ .cpu
```

```
                                                                 ↪ (),
                                                                 ↪ (1,2,0)
                                                                 ↪ ))
    plt.show()

optimizer.zero_grad() ## (61)
output = net(im_tensor) ## (62)
predictions_aug = output.view(self.rpg.batch_size,
    ↪ num_yolo_cells,num_anchor_boxes,9) ## (63)
loss = torch.tensor(0.0, requires_grad=True).float().to(self.
    ↪ rpg.device) ## (64)
for icx in range(num_yolo_cells): ## (65)
    for iax in range(num_anchor_boxes): ## (66)
        pred_yolo_vector = predictions_aug[:,icx,iax] ## (67)
        target_yolo_vector = yolo_tensor_aug[:,icx,iax] ##
            ↪ (68)
        ## Estiming presence/absence of object and the Binary
            ↪  Cross Entropy section:
        object_presence = nn.Sigmoid()(torch.unsqueeze(
            ↪ pred_yolo_vector[:,0], dim=0)) ## (69)
        target_for_prediction = torch.unsqueeze(
            ↪ target_yolo_vector[:,0], dim=0) ## (70)
        bceloss = criterion1(object_presence,
            ↪ target_for_prediction) ## (71)
        loss += bceloss ## (72)
        ## MSE section for regression params:
        pred_regression_vec = pred_yolo_vector[:,1:5] ## (73)
        pred_regression_vec = torch.unsqueeze(
            ↪ pred_regression_vec, dim=0) ## (74)
        target_regression_vec = torch.unsqueeze(
            ↪ target_yolo_vector[:,1:5], dim=0) ## (75)
        regression_loss = criterion2(pred_regression_vec,
            ↪ target_regression_vec) ## (76)
        loss += regression_loss ## (77)
        ## CrossEntropy section for object class label:
        probs_vector = pred_yolo_vector[:,5:] ## (78)
        probs_vector = torch.unsqueeze( probs_vector, dim=0 )
            ↪ ## (79)
        target = torch.argmax(target_yolo_vector[:,5:], dim=0)
            ↪  ## (80)
        target = torch.unsqueeze( target, dim=0 ) ## (81)
        class_labeling_loss = criterion3(probs_vector, target)
```

```
            ↪  ## (82)
        loss += class_labeling_loss ## (83)
if yolo_debug:
    print("\n\nshape␣of␣loss:␣", loss.shape)
    print("\n\nloss:␣", loss)
loss.backward() ## (84)
optimizer.step() ## (85)
running_loss += loss.item() ## (86)
if iter%15==14: ## (87)
    if display_images:
        print("\n\n\n") ## for vertical spacing for the image
            ↪  to be displayed later
    current_time = time.perf_counter()
    elapsed_time = current_time - start_time
    avg_loss = running_loss / float(1000) ## (88)
    # print("\n[epoch:%d/%d, iter=%4d elapsed_time=%5d secs]
        ↪  mean value for loss: %7.4f" %
                                        # (epoch+1,self.rpg.
                                           ↪ epochs, iter+1,
                                           ↪ elapsed_time,
                                           ↪ avg_loss)) ## (89)
    Loss_tally.append(running_loss)
    FILE1.write("%.3f\n" % avg_loss)
    FILE1.flush()
    running_loss = 0.0 ## (90)
    if display_labels:
        correct = 0
        total_objects = 0
        predictions = output.view(self.rpg.batch_size,
            ↪ num_yolo_cells,num_anchor_boxes,9) ## (91)
        if yolo_debug:
            print("\n\nyolo_vector␣for␣first␣image␣in␣batch,␣
                ↪ cell␣indexed␣18,␣and␣AB␣indexed␣2:␣")
            print(predictions[0, 18, 2])
        for ibx in range(predictions.shape[0]): # for each
            ↪ batch image ## (92)
            icx_2_best_anchor_box = {ic : None for ic in range
                ↪ (36)} ## (93)
            for icx in range(predictions.shape[1]): # for each
                ↪  yolo cell ## (94)
                cell_predi = predictions[ibx,icx] ## (95)
                prev_best = 0 ## (96)
                for anchor_bdx in range(cell_predi.shape[0]): #
                   ↪ # (97)
                    if cell_predi[anchor_bdx][0] > cell_predi[
                       ↪ prev_best][0]: ## (98)
                        prev_best = anchor_bdx ## (99)
                best_anchor_box_icx = prev_best ## (100)
                icx_2_best_anchor_box[icx] =
                   ↪ best_anchor_box_icx ## (101)
            sorted_icx_to_box = sorted(icx_2_best_anchor_box,
                    key=lambda x: predictions[ibx,x,
                       ↪ icx_2_best_anchor_box[x]][0].item(),
                       ↪ reverse=True) ## (102)
            retained_cells = sorted_icx_to_box[:5] ## (103)
            objects_detected = []
            pt = [] ## (104)
            for icx in retained_cells: ## (105)
                pred_vec = predictions[ibx,icx,
                   ↪ icx_2_best_anchor_box[icx]] ## (106)
                class_labels_predi = pred_vec[-4:] ## (107)
                class_labels_probs = torch.nn.Softmax(dim=0)(
                   ↪ class_labels_predi) ## (108)
                class_labels_probs = class_labels_probs[:-1] ##
                   ↪  (109)
                if torch.all(class_labels_probs < 0.25): ##
                   ↪ (110)
                    predicted_class_label = None
                    pt.append(13) ## (111)
                else:
                    best_predicted_class_index = (
                       ↪ class_labels_probs ==
                       ↪ class_labels_probs.max()) ## (112)
                    best_predicted_class_index =torch.nonzero(
                       ↪ best_predicted_class_index,as_tuple=
                       ↪ True)## (113)
                    predicted_class_label =self.rpg.
                       ↪ class_labels[
                       ↪ best_predicted_class_index[0].item()]
                       ↪  ## (114)
                    pt.append(best_predicted_class_index[0].
                       ↪ item())
                    objects_detected.append(
                       ↪ predicted_class_label) ## (115)
                    print("[batch␣image=%d]␣␣objects␣found␣in␣
                       ↪ descending␣probability␣order:␣" % ibx,
```

```
        gt = np.array(bbox_label_tensor[ibx,:].tolist())
        # pt = np.array(objects_detected)
        correct += np.sum(gt==pt)
        total_objects += np.sum(gt != None)
        acc = 100.0*correct/total_objects
    print("\n[epoch:%d/%d,␣iter=%4d␣␣elapsed_time=%5d␣secs
       ↪ ]␣␣␣␣␣␣mean␣value␣for␣loss:␣%7.4f␣␣␣␣␣Accuracy:␣
       ↪ %0.2f" %
              (epoch+1,self.rpg.epochs, iter+1,
                 ↪ elapsed_time, avg_loss, acc))
    if acc_max < acc:
        self.save_yolo_model(acc_max, acc, net, "/home/
           ↪ varun/work/courses/why2learn/hw/hw6/runs")
        acc_max = acc


    # print("Epoch %d Accuracy")
if display_images:
    logger = logging.getLogger()
    old_level = logger.level
    logger.setLevel(100)
    plt.figure(figsize=[15,4])
    plt.imshow(np.transpose(torchvision.utils.make_grid(
       ↪ im_tensor, normalize=True,
                                                  padding
                                                     ↪ =3,
                                                     ↪
                                                     ↪ pad_value
                                                     ↪ =255)
                                                     ↪ .
                                                     ↪ cpu
```

```
                                                                       ↪ ()
                                                                       ↪ ,
                                                                       ↪
                                                                       ↪ (1,2,0)
                                                                       ↪ )
                                                                       ↪ )
                      plt.show()
                      logger.setLevel(old_level)
       print("\nFinished␣Training\n")
       plt.figure(figsize=(10,5))
       plt.title("Loss␣vs.␣Iterations")
       plt.plot(Loss_tally)
       plt.xlabel("iterations")
       plt.ylabel("Loss")
       plt.legend()
       plt.savefig("training_loss.png")
       plt.show()
       torch.save(net.state_dict(), self.rpg.path_saved_yolo_model)
       return net

## set the dataloaders
# yolo.set_dataloaders(train=True)
# yolo.set_dataloaders(test=True)
yolo = batchedYOLO( rpg = rpg )

## custom dataloader
# prepare train dataloader
transform = tvt.Compose([tvt.ToTensor(),tvt.Normalize((0.5, 0.5, 0.5), (0.5,
   ↪  0.5, 0.5))])
# coco = COCO('/home/varun/work/courses/why2learn/hw/annotations/
   ↪ instances_train2017.json')
coco = []
dataserver_train = CocoDetection(transform, rpg.dataroot_train, rpg.
   ↪ class_labels, rpg.image_size, coco, loadDict=True, saveDict=False,
   ↪ mode="train")
yolo.train_dataloader = torch.utils.data.DataLoader(dataserver_train,
   ↪ batch_size=rpg.batch_size, shuffle=True, num_workers=8)

model = yolo.NetForYolo(skip_connections=True, depth=8)

number_of_learnable_params = sum(p.numel() for p in model.parameters() if p.
   ↪ requires_grad)
```

```
print("\n\nThe␣number␣of␣learnable␣parameters␣in␣the␣model:␣%d" %
   ↪ number_of_learnable_params)
num_layers = len(list(model.parameters()))
print("\n\nThe␣number␣of␣layers␣in␣the␣model:␣%d\n\n" % num_layers)

# train, test, both
mode = "train"
if mode=="train" or mode=="both":
    model = yolo.run_code_for_training_multi_instance_detection(model,
       ↪ display_images=False, display_labels=True)
if mode=="test" or mode=="both":
    yolo.run_code_for_testing_multi_instance_detection(model, display_images
       ↪  = True)
```

CODE-hw06_validation.py

```
from distutils.log import debug
import random, time
import numpy as np
import torch
import os, sys

from pycocotools.coco import COCO
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torch.optim as optim
import torchvision.transforms as tvt
import torchvision.transforms.functional as F
import torchvision.utils as tutils


from PIL import Image
from PIL import ImageDraw
from PIL import ImageTk
from PIL import ImageFont

import sys,os,os.path,glob,signal
import re
import functools
import math
import random
import copy
import gzip
import pickle

if sys.version_info[0] == 3:
    import tkinter as Tkinter
    from tkinter.constants import *
else:
    import Tkinter
    from Tkconstants import *

import matplotlib.pyplot as plt
import logging

# seed = 0
# random.seed(seed)
# torch.manual_seed(seed)
# torch.cuda.manual_seed(seed)
# numpy.random.seed(seed)
# torch.backends.cudnn.deterministic=True
# torch.backends.cudnn.benchmarks=False
# os.environ[’PYTHONHASHSEED’] = str(seed)

# USER imports
sys.path.append("/home/varun/work/courses/why2learn/hw/RPG-2.0.6")
from RegionProposalGenerator import *

from model import pneumaNet
from dataloader import CocoDetection
import utils


rpg = RegionProposalGenerator(
                dataroot_train = "/home/varun/work/courses/why2learn/hw/hw6
                    ↪ /data/",
                dataroot_test = "/home/varun/work/courses/why2learn/hw/hw6/
                    ↪ data/",
                image_size = [120,120],
                yolo_interval = 20,
                path_saved_yolo_model = "/home/varun/work/courses/why2learn
```

```
                      ↪ /saved_yolo_model_lr-4_bs_71_depth16.pt",
                  momentum = 0.5,
                  learning_rate = 1e-4,
                  epochs = 50,
                  batch_size = 1,
                  classes = ['car','motorcycle','stop␣sign'],
                  use_gpu = True,
              )

class batchedYOLO(RegionProposalGenerator.YoloLikeDetector):
    def __init__(self, rpg):
        super().__init__(rpg)

    def save_yolo_model(self, acc_max, acc, model, path):
        if acc_max != 0:
            oldSaveName = "model_" + str(self.rpg.epochs) + "_" + str(self.
                ↪ rpg.learning_rate) + "_" + str(self.rpg.batch_size) + "_"
                ↪ + str(model.depth) +"_" + str(np.round(acc_max,2)) + "
                ↪ _best.pt"
            oldSaveName = os.path.join(path,oldSaveName)
            os.remove(oldSaveName)
        saveName = "model_" + str(self.rpg.epochs) + "_" + str(self.rpg.
            ↪ learning_rate) + "_" + str(self.rpg.batch_size) + "_" + str(
            ↪ model.depth) +"_" + str(np.round(acc,2)) + "_best.pt"
        saveName = os.path.join(path,saveName)
        torch.save(model, saveName)

    def run_code_for_testing_multi_instance_detection(self, net,
        ↪ display_labels=False, display_images=False):
        # net.load_state_dict(torch.load(self.rpg.path_saved_yolo_model))
        net = net.to(self.rpg.device)
        yolo_interval = self.rpg.yolo_interval
        num_yolo_cells = (self.rpg.image_size[0] // yolo_interval) * (self.
            ↪ rpg.image_size[1] // yolo_interval)
        num_anchor_boxes = 5 # (height/width) 1/5 1/3 1/1 3/1 5/1
        yolo_tensor = torch.zeros( self.rpg.batch_size, num_yolo_cells,
            ↪ num_anchor_boxes, 8 )
        with torch.no_grad():
            for iter, data in enumerate(self.test_dataloader):
                im_tensor, seg_mask_tensor, bbox_tensor, bbox_label_tensor,
                    ↪ num_objects_in_image = data
                if iter % 5 == 4:
```

```
print("\n\n\n\nShowing␣output␣for␣test␣batch␣%d:␣" % (iter
    ↪ +1))
im_tensor = im_tensor.to(self.rpg.device)
seg_mask_tensor = seg_mask_tensor.to(self.rpg.device)
bbox_tensor = bbox_tensor.to(self.rpg.device)
bbox_label_tensor = bbox_label_tensor.to(self.rpg.device)
yolo_tensor = yolo_tensor.to(self.rpg.device)

output = net(im_tensor)
predictions = output.view(self.rpg.batch_size,
    ↪ num_yolo_cells,num_anchor_boxes,9)
for ibx in range(predictions.shape[0]): # for each batch
    ↪ image
    icx_2_best_anchor_box = {ic : None for ic in range(36)
        ↪ }
    for icx in range(predictions.shape[1]): # for each
        ↪ yolo cell
        cell_predi = predictions[ibx,icx]
        prev_best = 0
        for anchor_bdx in range(cell_predi.shape[0]): #
            ↪ for each anchor box
            if cell_predi[anchor_bdx][0] > cell_predi[
                ↪ prev_best][0]:
                prev_best = anchor_bdx
        best_anchor_box_icx = prev_best
        icx_2_best_anchor_box[icx] = best_anchor_box_icx
    sorted_icx_to_box = sorted(icx_2_best_anchor_box,
                key=lambda x: predictions[ibx,x,
                    ↪ icx_2_best_anchor_box[x]][0].item(),
                    ↪ reverse=True)
    retained_cells = sorted_icx_to_box[:5]
objects_detected = []
pt = []
correct = 0
total_objects = 0
for icx in retained_cells:
    pred_vec = predictions[ibx,icx, icx_2_best_anchor_box[
        ↪ icx]]
    class_labels_predi = pred_vec[-4:]
    class_labels_probs = torch.nn.Softmax(dim=0)(
        ↪ class_labels_predi)
    class_labels_probs = class_labels_probs[:-1]
```

```
                    if torch.all(class_labels_probs < 0.2):
                        predicted_class_label = None
                    else:
                        best_predicted_class_index = (class_labels_probs
                            ↪ == class_labels_probs.max())
                        pt.append(best_predicted_class_index[0].item())
                        best_predicted_class_index = torch.nonzero(
                            ↪ best_predicted_class_index, as_tuple=True)
                        predicted_class_label = self.rpg.class_labels[
                            ↪ best_predicted_class_index[0].item()]
                        objects_detected.append(predicted_class_label)
                    gt = np.array(bbox_label_tensor[ibx,:].tolist())
                    correct += np.sum(gt==pt)
                    total_objects += np.sum(gt != 13)
                    acc = 100.0*correct/total_objects
                print("[batch␣image=%d]␣␣objects␣found␣in␣descending␣
                    ↪ probability␣order:␣" % ibx, objects_detected)
                print("[batch␣image=%d]␣␣objects␣present:␣" % ibx,
                    ↪ bbox_label_tensor[ibx,:])

                print("Acc:␣%0.2f"%(acc))
                logger = logging.getLogger()
                old_level = logger.level
                logger.setLevel(100)
                if display_images:
                    plt.figure(figsize=[15,4])
                    plt.imshow(np.transpose(torchvision.utils.make_grid(
                        ↪ im_tensor, normalize=True,
                                                                 padding=3,
                                                                     ↪ pad_value
                                                                     ↪ =255)
                                                                     ↪ .cpu
                                                                     ↪ (),
                                                                     ↪ (1,2,0)
                                                                     ↪ ))
                    plt.show()
                logger.setLevel(old_level)

## set the dataloaders
# yolo.set_dataloaders(train=True)
# yolo.set_dataloaders(test=True)
yolo = batchedYOLO( rpg = rpg )
```

```
## custom dataloader
# prepare train dataloader
transform = tvt.Compose([tvt.ToTensor(),tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
# coco = COCO('/home/varun/work/courses/why2learn/hw/annotations/instances_train2017.json')
coco = []
dataserver_test = CocoDetection(transform, rpg.dataroot_test, rpg.class_labels, rpg.image_size, coco, loadDict=True, saveDict=False, mode="test")
yolo.test_dataloader = torch.utils.data.DataLoader(dataserver_test, batch_size=rpg.batch_size, shuffle=True, num_workers=8)

# model = yolo.NetForYolo(skip_connections=True, depth=8)
model = torch.load("/home/varun/work/courses/why2learn/hw/hw6/runs/model_50_0.0001_71_4_38.31_best.pt")

number_of_learnable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print("\n\nThe number of learnable parameters in the model: %d" % number_of_learnable_params)
num_layers = len(list(model.parameters()))
print("\n\nThe number of layers in the model: %d\n\n" % num_layers)

# train, test, both
mode = "test"
if mode=="train" or mode=="both":
    model = yolo.run_code_for_training_multi_instance_detection(model, display_images=False, display_labels=True)
if mode=="test" or mode=="both":
    yolo.run_code_for_testing_multi_instance_detection(model, display_images = True)
```

CODE-dataloader.py

```
"""
Homework 6: Implement part of YOLO logic

Author: Varun Aggarwal
Last Modified: 28 Mar 2022
"""
```

```
from pycocotools.coco import COCO
import os
import numpy as np
import skimage.io as io
import matplotlib.pyplot as plt
import cv2

import torch
from torch.utils.data import Dataset
from PIL import Image
import os, glob, sys
import numpy as np
import utils, pickle

class CocoDetection(Dataset):
    def __init__(self,transform,dataPath,classes,size,coco,loadDict=False,
       ↪ saveDict=False,mode="test"):
        self.dataPath = dataPath
        self.class_labels = classes
        self.transform = transform

        # load pre-existing dictionary
        if loadDict and mode=="train":
            with open(os.path.join(dataPath,’dictTrain.pkl’), ’rb’) as file:
                self.imgDict = pickle.load(file)
        elif loadDict and mode=="test":
            with open(os.path.join(dataPath,’dictTest.pkl’), ’rb’) as file:
                self.imgDict = pickle.load(file)
        elif mode=="train":
            self.imgDict = utils.downloadCOCO(dataPath,classes,size,coco,
               ↪ saveDict)
        elif mode=="test":
            self.imgDict = utils.downloadCOCO(dataPath,classes,size,coco,
               ↪ saveDict)
        else:
            print("Something␣is␣wrong␣here␣!!!")
            sys.exit()
        self.imgDict = list(self.imgDict.values())

    def __len__(self):
        return len(self.imgDict)
    def __getitem__(self,idx):
        img = self.imgDict[idx]
        bbox_label = img['bbox_label']
        bbox = img['bbox']
        noObjects = torch.tensor(img['no_of_objects'], dtype=torch.long)
        if self.transform:
            image = self.transform(img['imgActual'])
        # im_tensor, segmentation, bbox, label, numberOfObjects
        return image, torch.zeros(1, dtype=torch.uint8), bbox, bbox_label,
           ↪ noObjects
```

CODE-utils.py

```
"""
Homework 6: Implement part of YOLO logic

Author: Varun Aggarwal
Last Modified: 28 Mar 2022
"""

from pycocotools.coco import COCO
import os, sys
import numpy as np
import skimage.io as io
import matplotlib.pyplot as plt
import itertools

from PIL import Image
import requests
from io import BytesIO
import pickle
import torch

def normalize(data,min=0.0,max=127.0):
    return list((np.array(data) - min)/(max-min))

def unnormalize(data,min=0.0,max=127.0):
    return list(np.array(data)*(max-min)+min)

# funciton for checking if img follows the spec
def checkImgAnn(root_path, classes, img, catIds, anns, size, coco):
```

```
filePath = os.path.join(root_path, img['file_name'])
if os.path.exists(filePath):
    imgActual = Image.open(filePath)
else:
    response = requests.get(img['coco_url'])
    imgActual = Image.open(BytesIO(response.content))
    imgActual = imgActual.convert(mode='RGB')
    imgActual.save(filePath)

# bbox scaling parameters
w,h = imgActual.size
wScale, hScale = size[0]/w, size[1]/h

objPresenceCount = len(anns)
# choose first five annotations it total is over 5
if objPresenceCount > 5:
    anns = anns[0:5]
    objPresenceCount = 5

# prepare bbox and label tensor
bbox_tensor = torch.zeros(5,4, dtype=torch.uint8)
bbox_label_tensor = torch.zeros(5, dtype=torch.uint8) + 13 # for empty
   ↪ object
for i in range(objPresenceCount):
    ## normalize bbox
    wScale, hScale = size[0]/w, size[1]/h
    bbox = anns[i]['bbox']
    bbox = [wScale*(bbox[0]),hScale*(bbox[1]),wScale*(bbox[0]+bbox[2]-1)
       ↪ ,hScale*(bbox[1]+bbox[3]-1)]
    # can improve the logic here for seperate width and height
    for j,coor in enumerate(bbox):
        if coor > size[0]-1:
            bbox[j] = size[0]-1
        elif coor < 0:
            bbox[j] = 0
    # bbox = normalize(bbox,0,123)
    # save bbox and label in tensor
    bbox_tensor[i] = torch.LongTensor(bbox)
    bbox_label_tensor[i] = catIds.index(anns[i]['category_id'])

## For valid images, return image dictionary
imgDict = {}
```

```
    imgDict['bbox_label'] = bbox_label_tensor
    imgDict['no_of_objects'] = objPresenceCount
    imgDict['bbox'] = bbox_tensor
    imgDict['imgActual'] = imgActual.resize(size)
    return {filePath:imgDict}


# coco - instance of COCO class -> coco = COCO(jsonPath)
def downloadCOCO(root_path, classes, size=(128,128), coco=None, saveDict=
   ↪ False, mode="test"):
    # create a dictionary of images which can be used for training/
       ↪ validation
    dictImgs = {}
    # check if root image folder already exists if not then create one
    if not os.path.exists(root_path):
        os.makedirs(root_path)


    # get category ids for all classes
    catIds = coco.getCatIds(catNms=classes)
    imgIds = []

    # get images with atleast two different clases
    for data in itertools.combinations(catIds,2):
        imgIds.extend(coco.getImgIds(catIds=data))
    imgIds = list(set(imgIds))

    imgs = coco.loadImgs(imgIds)

    # Start Downloading
    print("Downloading␣%d␣images␣"%(len(imgIds)))
    for img in imgs:
        # get all annotations for the img
        annIds = coco.getAnnIds(imgIds=img['id'], catIds=catIds)
        anns = coco.loadAnns(annIds)
        # anns = [x for x in anns if x['category_id'] in catIds]

        # download only if annotation are to the spec i.e. less than 5 anns
        imgDict = checkImgAnn(root_path, classes, img, catIds, anns, size,
           ↪ coco)

        if imgDict != False:
            dictImgs.update(imgDict)

    # save the dictionary for faster access
    if saveDict==True and mode=="train":
        with open (os.path.join(root_path, 'dictTrain.pkl'),'wb') as file:
            pickle.dump(dictImgs, file)
    elif saveDict==True and mode=="test":
        with open (os.path.join(root_path, 'dictTest.pkl'),'wb') as file:
            pickle.dump(dictImgs, file)
    return dictImgs

# coco = COCO('/home/varun/work/courses/why2learn/hw/annotations/instances_train2017.json')
# dictImgs = downloadCOCO('/home/varun/work/courses/why2learn/hw/hw6/data',['car','motorcycle','stop sign'],(128,128),coco, True, "train")
# coco = COCO('/home/varun/work/courses/why2learn/hw/annotations/instances_val2017.json')
# dictImgs = downloadCOCO('/home/varun/work/courses/why2learn/hw/hw6/data',['car','motorcycle','stop sign'],(128,128),coco, True, "test")
# print("The size of the dictionary is {} bytes".format(sys.getsizeof(dictImgs)))
# [x for x in temp4 if x['category_id'] in [3,13]]
# print()
```

4 Lessons Learned

The programming homework was straightforward and covered the basics of YOLO implementation for object detection. Although, the programming took a while to complete, I did not find any major issues with implementation.

5 Suggested Enhancements

The direction for the assignment could be clarified in advance. Maybe even, students can be asked to implement the YOLOv1 network and datalaoder by themselves instead of relying on code from RPG2.0.6