ECE695DL: Homework 6

Tal A. Bass

14 April 2021

hw06_training.py

```
'''
hw06_training.py

Tal A. Bass
Deep Learning BME965/ECE695
Homework 6
Spring 2021
Due 04/14/2021

Some code was copied from RegionProposalGenerator, including the AnchorBox class, the logic for
↪  assigning the anchorbox, and creating the yolo vector. All other code is my own work. Much of the
↪  code that was copied over had to be adjusted for my format.

This code represents my own work in accordance with the academic integrity policies of Purdue
↪  University.

Resources referenced:
Course lecture notes - https://engineering.purdue.edu/DeepLearn/
RegionProposalGenerator - https://engineering.purdue.edu/kak/distRPG/RegionProposalGenerator-2.0.1.html

'''

import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset
import glob
import torchvision.transforms as tvt
import random
import os
from PIL import Image
import matplotlib.pyplot as plt
from dataloader import Image_Dataset
from model import Net
import torch.optim as optim
```

```
# Copied directly from course notes to get reproducible results
seed = 0
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
np.random.seed(seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmarks=False
os.environ['PYTHONHASHSEED'] = str(seed)

class AnchorBox:
    def __init__(self, AR, tlc, ab_height, ab_width):         # toc : top_left_corner      ab : anchor
    ↪  box
        self.AR = AR
        self.tlc = tlc
        self.ab_height = ab_height
        self.ab_width = ab_width

# Scripted arguments
root_path = "C:/Users/pavlo/Desktop/Tal Purdue Classes/Deep Learning/ImageDatasets/hw06/Train/"
anns_folder = root_path + 'annotations/'
ann_path = anns_folder + 'my_annotations.p'
class_list = ["bus", "truck", "car"]
class_labels = class_list

# Define number of epochs
epochs = 150
learning_rate = 0.5e-4
momentum = 0.9
my_batch_size = 5

transform = tvt.Compose([tvt.ToTensor() , tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
train_dataset = Image_Dataset(root_path, ann_path, class_list, transform)
train_data_loader = torch.utils.data.DataLoader(dataset = train_dataset, batch_size = my_batch_size,
↪  shuffle = True, num_workers = 4)

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Using float32 to be compatible with the dataset which is also saved in float32
dtype = torch.float32

net = Net()
net = net.to(device)
cross_entropy_criterion = nn.CrossEntropyLoss()
mse_criterion = nn.MSELoss()
optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum=momentum)
loss_record = []

# YOLO parameters
yolo_interval = 20
image_size = [128, 128]
```

```
num_yolo_cells = 36
num_anchor_boxes =  5    # (height/width)   1/5  1/3  1/1  3/1  5/1
#  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 delta diffs between the center
#  of the yolo cell and the center of the object bounding box in terms of
#  a unit for the cell.  bh and bw are the height and the width of the
#  object bounding box in terms of the cell heidht and width.

for epoch in range(epochs):
    running_loss = 0.0
    for i, data in enumerate(train_data_loader):
        yolo_tensor = torch.zeros(my_batch_size, num_yolo_cells, num_anchor_boxes, 8) # Need to zero
        ↪  out the yolo_tensor for each new batch

        im_tensor, bbox_tensor, bbox_label_tensor, num_objects_in_image = data
        im_tensor = im_tensor.to(device)
        bbox_tensor = bbox_tensor.to(device)
        bbox_label_tensor = bbox_label_tensor.to(device)
        yolo_tensor = yolo_tensor.to(device)
        cell_height = yolo_interval
        cell_width = yolo_interval
        num_cells_image_width = image_size[0] // yolo_interval
        num_cells_image_height = image_size[1] // yolo_interval

        # Construct the anchor boxes
        anchor_boxes_1_1 = [[AnchorBox("1/1", (i*yolo_interval,j*yolo_interval), yolo_interval,
        ↪  yolo_interval)
                                                              for i in range(0,num_cells_image_height)]
                                                                 for j in range(0,num_cells_image_width)]
        anchor_boxes_1_3 = [[AnchorBox("1/3",(i*yolo_interval,j*yolo_interval), yolo_interval,
        ↪  3*yolo_interval)
                                                              for i in range(0,num_cells_image_height)]
                                                                 for j in range(0,num_cells_image_width)]
        anchor_boxes_3_1 = [[AnchorBox("3/1",(i*yolo_interval,j*yolo_interval), 3*yolo_interval,
        ↪  yolo_interval)
                                                              for i in range(0,num_cells_image_height)]
                                                                 for j in range(0,num_cells_image_width)]
        anchor_boxes_1_5 = [[AnchorBox("1/5",(i*yolo_interval,j*yolo_interval), yolo_interval,
        ↪  5*yolo_interval)
                                                              for i in range(0,num_cells_image_height)]
                                                                 for j in range(0,num_cells_image_width)]
        anchor_boxes_5_1 = [[AnchorBox("5/1",(i*yolo_interval,j*yolo_interval), 5*yolo_interval,
        ↪  yolo_interval)
                                                              for i in range(0,num_cells_image_height)]
                                                                 for j in range(0,num_cells_image_width)]

        for ibx in range(im_tensor.shape[0]): # For each image in the batch
                        for idx in range(num_objects_in_image[ibx]): # For each object in the image
                            # Note that the bbox coordinates are [x,y,width,height]
```

```
height_center_bb =  bbox_tensor[ibx][idx][1].item() +
↪  (bbox_tensor[ibx][idx][3].item() // 2) # y + height/2
width_center_bb =  bbox_tensor[ibx][idx][0].item() +
↪  (bbox_tensor[ibx][idx][2].item() // 2) # x _ width/2
obj_bb_height = bbox_tensor[ibx][idx][3].item()
obj_bb_width = bbox_tensor[ibx][idx][2].item()

if (obj_bb_height < 4) or (obj_bb_width < 4):
    continue
AR = float(obj_bb_height) / float(obj_bb_width)

cell_row_idx =  height_center_bb // yolo_interval          ## for the i
↪  coordinate
cell_col_idx = width_center_bb // yolo_interval            ## for the j
↪  coordinates
cell_row_idx = 5 if cell_row_idx > 5 else cell_row_idx
cell_col_idx = 5 if cell_col_idx > 5 else cell_col_idx

if AR <= 0.2:
    anchbox = anchor_boxes_1_5[cell_row_idx][cell_col_idx]
    anchor_number = 0
elif AR <= 0.5:
    anchbox = anchor_boxes_1_3[cell_row_idx][cell_col_idx]
    anchor_number = 1
elif AR <= 1.5:
    anchbox = anchor_boxes_1_1[cell_row_idx][cell_col_idx]
    anchor_number = 2
elif AR <= 4:
    anchbox = anchor_boxes_3_1[cell_row_idx][cell_col_idx]
    anchor_number = 3
elif AR > 4:
    anchbox = anchor_boxes_5_1[cell_row_idx][cell_col_idx]
    anchor_number = 4

# bh and bw calculations
# bh is the height of the bounding-box divided by the actual height of the
↪  grid cell
bh  =  100*float(obj_bb_height) / float(yolo_interval)
bw  =  100*float(obj_bb_width)  / float(yolo_interval)

# bx, by calculations (bx is del_x, by is del_y)
obj_center_x =  float(bbox_tensor[ibx][idx][2].item() +
↪  (bbox_tensor[ibx][idx][0].item()/2.0))
obj_center_y =  float(bbox_tensor[ibx][idx][3].item() +
↪  (bbox_tensor[ibx][idx][1].item()/2.0))
yolocell_center_i =  cell_row_idx*yolo_interval + float(yolo_interval) /
↪  2.0
yolocell_center_j =  cell_col_idx*yolo_interval + float(yolo_interval) /
↪  2.0
del_x = float(obj_center_x - yolocell_center_j) / yolo_interval
del_y = float(obj_center_y - yolocell_center_i) / yolo_interval
```

```
                            yolo_vector = [1, del_x, del_y, bh, bw, 0, 0, 0]

                            # class label
                            #label_index =
                            ↪  (bbox_label_tensor[ibx][idx].nonzero(as_tuple=True)[0]).item()
                            #label = class_labels[label_index]
                            yolo_vector[5 + bbox_label_tensor[ibx][idx].item()] = 1
                            yolo_cell_index =  cell_row_idx * num_cells_image_width  +  cell_col_idx
                            yolo_tensor[ibx, yolo_cell_index, anchor_number] = torch.FloatTensor(
                            ↪  yolo_vector )
        yolo_tensor_flattened = yolo_tensor.view(im_tensor.shape[0], -1)
        optimizer.zero_grad()
        output = net(im_tensor)
        loss = mse_criterion(output, yolo_tensor_flattened)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i % 100 == 99:
            avg_loss = running_loss / float(99)
            print("\n[epoch:%d, iteration:%5d]  loss: %.3f  " % (epoch + 1, i + 1, avg_loss))
            loss_record.append(avg_loss)
            running_loss = 0.0

plt.figure()
plt.plot(loss_record)
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.savefig("train_loss.jpg")

torch.save(net.state_dict(), "net.pth")
```

hw06_validation.py

```
'''
hw06_validation.py

Tal A. Bass
Deep Learning BME965/ECE695
Homework 6
Spring 2021
Due 04/14/2021

Some code for decoding the yolo vector was copied from RegionProposalGenerator and heavily adjusted.
↪  All other code is my own work.

This code represents my own work in accordance with the academic integrity policies of Purdue
↪  University.
```

```
Resources referenced:
Course lecture notes - https://engineering.purdue.edu/DeepLearn/
RegionProposalGenerator - https://engineering.purdue.edu/kak/distRPG/RegionProposalGenerator-2.0.1.html

'''

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset
import glob
import numpy as np
import torchvision.transforms as tvt
import torchvision
import random
import os
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')
from dataloader import Image_Dataset
from model import Net
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import cv2

# Copied directly from course notes to get reproducible results
seed = 0
random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
np.random.seed(seed)
torch.backends.cudnn.deterministic=True
torch.backends.cudnn.benchmarks=False
os.environ['PYTHONHASHSEED'] = str(seed)

# Scripted arguments
root_path = "C:/Users/pavlo/Desktop/Tal Purdue Classes/Deep Learning/ImageDatasets/hw06/Val/"
anns_folder = root_path + 'annotations/'
ann_path = anns_folder + 'my_annotations.p'
class_list = ["bus", "chair", "car"]
class_labels = class_list

show_imgs = True

my_batch_size = 1
yolo_interval = 20
image_size = [128, 128]
num_yolo_cells = 36
num_anchor_boxes =  5
max_objects = 10
```

```
# Create a list of cell bboxs for reference to use with false alarms
num_cells_image_width = image_size[0] // yolo_interval
num_cells_image_height = image_size[1] // yolo_interval
cell_bboxs = [0]*num_yolo_cells
for i in range(0,num_cells_image_height):
   for j in range(0,num_cells_image_width):
        cell_bboxs[i+6*j] = [i*yolo_interval, j*yolo_interval, yolo_interval, yolo_interval]

all_labels = []
all_pred = []
iou_record = []
false_alarm_car_record = 0
false_alarm_truck_record = 0
false_alarm_bus_record = 0
false_alarm_record = 0
true_alarm_record = 0

def intersection_over_union(bbox1_uint8, bbox2_uint8):
    bbox1 = [int(bbox1_uint8[0]),int(bbox1_uint8[1]),int(bbox1_uint8[2]),int(bbox1_uint8[3])]
    bbox2 = [int(bbox2_uint8[0]),int(bbox2_uint8[1]),int(bbox2_uint8[2]),int(bbox2_uint8[3])]

    # Which one is on the left
    if bbox1[0] < bbox2[0]:
        leftbox = bbox1
        rightbox = bbox2
    else:
        leftbox = bbox2
        rightbox = bbox1
    # x overlap
    x_overlap = leftbox[0] + leftbox[2] - rightbox[0]
    if ((leftbox[0] + leftbox[2])>(rightbox[0] + rightbox[2])):
        x_overlap -= ((leftbox[0] + leftbox[2])-(rightbox[0] + rightbox[2]))
    if x_overlap < 0:
        x_overlap = 0
    # Which one is higher
    if bbox1[1] < bbox2[1]:
        highbox = bbox1
        lowbox = bbox2
    else:
        highbox = bbox2
        lowbox = bbox1
    # y overlap
    y_overlap = highbox[1] + highbox[3] - lowbox[1]
    if ((highbox[1] + highbox[3])>(lowbox[1] + lowbox[3])):
        y_overlap -= ((highbox[1] + highbox[3])-(lowbox[1] + lowbox[3]))
    if y_overlap < 0:
        y_overlap = 0

    intersection = x_overlap * y_overlap
    union = ((bbox1[2]*bbox1[3])+(bbox2[2]*bbox2[3])-intersection)
```

```
    if (union) :
        iou = intersection/union
    else: # Prevent divide by 0
        iou = 0
    return iou

# Takes in a cell number and offsets [bx, by, bh, bw] to [x, y, h, w]
def icx_offset_to_bbox(icx, offsets):
    cell_bbox = cell_bboxs[icx]
    h = yolo_interval*offsets[2]
    w = yolo_interval*offsets[3]
    x_center = cell_bbox[0]+(yolo_interval//2)
    y_center = cell_bbox[1]+(yolo_interval//2)
    x = x_center + offsets[0] - h//2
    y = y_center + offsets[1] - w//2
    return [x, y, h, w]

def find_best_match(pred_bbox, bbox_tensor):
    bbox_array = bbox_tensor.numpy()
    best_iou = 0
    best_idx = 0
    for idx in range(bbox_array.shape[0]):
        cur_iou = intersection_over_union(pred_bbox, bbox_array[idx,:])
        if (cur_iou > best_iou):
            best_idx = idx
            best_iou = cur_iou
    return best_idx, best_iou

def plot_cfm(all_labels, all_pred, acc):
    cfm = confusion_matrix(all_labels, all_pred)
    disp = ConfusionMatrixDisplay(cfm,display_labels=class_list)
    disp = disp.plot(cmap=plt.cm.Blues, values_format = 'd')
    plt.xticks(rotation = 45)
    disp.ax_.set_title('Confusion Matrix \n Accuracy = %0.3f'%(acc))
    plt.savefig("net_confusion_matrix.jpg")
    plt.show()

def show_img(im_tensor, bbox_tensor, pred_tensor, bbox_label_tensor, num_objects_in_image):
    im_with_bbox_tensor = torch.clone(im_tensor)
    for i in range(num_objects_in_image):
        ii = bbox_tensor[i][0].item()
        ji = bbox_tensor[i][1].item()
        ki = (bbox_tensor[i][2].item())+ii
        li = (bbox_tensor[i][3].item())+ji
        if (ii > 127):
            ii = 127
        if (ji > 127):
            ji = 127
        if (ki > 127):
            ki = 127
        if (li > 127):
```

```
            li = 127
        im_with_bbox_tensor[:,ji,ii:ki] = 255
        im_with_bbox_tensor[:,li,ii:ki] = 255
        im_with_bbox_tensor[:,ji:li,ii] = 255
        im_with_bbox_tensor[:,ji:li,ki] = 255

        ii = pred_tensor[i][0].item()
        ji = pred_tensor[i][1].item()
        ki = (pred_tensor[i][2].item())+ii
        li = (pred_tensor[i][3].item())+ji
        if (ii > 127):
            ii = 127
        if (ji > 127):
            ji = 127
        if (ki > 127):
            ki = 127
        if (li > 127):
            li = 127
        im_with_bbox_tensor[0,ji,ii:ki] = 0.1
        im_with_bbox_tensor[0,li,ii:ki] = 0.1
        im_with_bbox_tensor[0,ji:li,ii] = 0.1
        im_with_bbox_tensor[0,ji:li,ki] = 0.1
        im_with_bbox_tensor[1,ji,ii:ki] = 0.8
        im_with_bbox_tensor[1,li,ii:ki] = 0.8
        im_with_bbox_tensor[1,ji:li,ii] = 0.8
        im_with_bbox_tensor[1,ji:li,ki] = 0.8
        im_with_bbox_tensor[2,ji,ii:ki] = 0.1
        im_with_bbox_tensor[2,li,ii:ki] = 0.1
        im_with_bbox_tensor[2,ji:li,ii] = 0.1
        im_with_bbox_tensor[2,ji:li,ki] = 0.1

    plt.figure(figsize=[5,5])
    plt.imshow(np.transpose(torchvision.utils.make_grid(im_with_bbox_tensor, normalize=False,
                                                        padding=3, pad_value=255).cpu(), (1,2,0)))
    plt.axis('tight')
    plt.axis('off')
    plt.show()
    #img_file_path = "C:/Users/pavlo/Desktop/Tal Purdue Classes/Deep Learning/HW
    ↪  Submissions/HW6/Validation_Pics/test.jpg"
    #im_resized.save(img_file_path)

def run_code_for_testing(net, path_saved_model):
    global all_labels
    global all_pred
    global false_alarm_car_record
    global false_alarm_truck_record
    global false_alarm_bus_record
    global false_alarm_record
    global true_alarm_record
    global iou_record
```

```
net.load_state_dict(torch.load(path_saved_model))
net = net.eval()
net.to(device)
with torch.no_grad():
    for i,data in enumerate(test_data_loader):
        im_tensor, bbox_tensor, bbox_label_tensor, num_objects_in_image = data
        im_tensor = im_tensor.to(device)
        bbox_tensor = bbox_tensor.to(device)
        bbox_label_tensor = bbox_label_tensor.to(device)
        outputs = net(im_tensor)

        predictions = outputs.view(my_batch_size,num_yolo_cells,num_anchor_boxes,8)

        # For each image in the batch
        for ibx in range(predictions.shape[0]):
            icx_2_best_anchor_box = {ic : None for ic in range(num_yolo_cells)}
            # For each yolo cell, find the best anchor box
            for icx in range(predictions.shape[1]):
                cell_predi = predictions[ibx,icx]
                prev_best = 0
                for anchor_bdx in range(cell_predi.shape[0]):
                    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)
            # Retain the cells with the 10 best anchor boxes
            retained_cells = sorted_icx_to_box[:max_objects]
            objects_detected = []
            pred_bbox_tensor = torch.zeros(max_objects,4,dtype=torch.uint8)
            pred_bbox_tensor_idx = 0
            for icx in retained_cells:
                pred_vec = predictions[ibx,icx, icx_2_best_anchor_box[icx]]
                class_labels_predi  = pred_vec[-3:]
                if torch.all(class_labels_predi < 0.05): # If none of the values are greater than
                ↪  0.05, there is no object
                    predicted_class_label = None
                else:
                    best_predicted_class_index = (class_labels_predi ==
                    ↪  class_labels_predi.max()).nonzero().squeeze().item()
                    predicted_class_label = class_labels[best_predicted_class_index]
                    objects_detected.append(predicted_class_label)

                    # False alarm checking
                    pred_bbox = icx_offset_to_bbox(icx, pred_vec[1:5])
                    # Make sure all values are positive
                    pred_bbox =  [abs(item) for item in pred_bbox]
                    # Make sure elements are not out of bounds
                    for idx,elem in enumerate(pred_bbox):
```

```
                        if pred_bbox[idx].item() > 255:
                            pred_bbox[idx] = torch.tensor(255)
                    pred_bbox_tensor[pred_bbox_tensor_idx,0] = pred_bbox[0].item()
                    pred_bbox_tensor[pred_bbox_tensor_idx,1] = pred_bbox[1].item()
                    pred_bbox_tensor[pred_bbox_tensor_idx,2] = pred_bbox[2].item()
                    pred_bbox_tensor[pred_bbox_tensor_idx,3] = pred_bbox[3].item()
                    pred_bbox_tensor_idx += 1
                    best_idx, best_iou = find_best_match(pred_bbox, bbox_tensor[ibx])
                    # If there is no overlap (iou = 0) with predicted bbox and bbox_tensor, false
                    ↪ alarm
                    if (best_iou):
                        all_labels.append(bbox_label_tensor[ibx][best_idx].item())
                        all_pred.append(best_predicted_class_index)
                        true_alarm_record += 1
                        iou_record.append(best_iou)
                    else:
                        false_alarm_record += 1
                        if (predicted_class_label == "bus"):
                            false_alarm_bus_record += 1
                        elif (predicted_class_label == "car"):
                            false_alarm_car_record += 1
                        elif (predicted_class_label == "truck"):
                            false_alarm_truck_record += 1

                if (show_imgs):
                    show_img(im_tensor[ibx], bbox_tensor[ibx], pred_bbox_tensor,
                    ↪ bbox_label_tensor[ibx], num_objects_in_image[ibx])


    matches = [i for i, j in zip(all_labels, all_pred) if i == j]
    acc = len(matches) / len(all_labels)
    #plot_cfm(all_labels, all_pred, acc)

transform = tvt.Compose([tvt.ToTensor() , tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
test_dataset = Image_Dataset(root_path, ann_path, class_list, transform)
test_data_loader = torch.utils.data.DataLoader(dataset = test_dataset, batch_size = my_batch_size,
↪ shuffle = True, num_workers = 0)

# Using float32 to be compatible with the dataset which is also saved in float32
dtype = torch.float32
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

run_code_for_testing(Net(), 'net.pth')

false_alarm_rate = false_alarm_record/(false_alarm_record+true_alarm_record)
print("False alarm rate: " + str(false_alarm_rate))

iou_avg = sum(iou_record)/len(iou_record)
print("Average Intersection Over Union: " + str(iou_avg))
```

HW06 Report

TAL BASS

Image Dataset/Dataloader

The image dataset consists of 600 pictures of cars, trucks, and buses. The image downloader only downloads an image if there is more than one bbox in the annotation (essentially if there is more than one object in the picture).

When the script downloads an image, it adds the annotations of that image to a JSON file which is then pickled. The key for each image is the filename.

Upon initialization, the dataloader loads the pickle file and saves all of the annotations. Each item in the dataset returns the image, the bboxs, a corresponding tensor of labels, and the number of objects in the image. The max number of objects for any image is 10.

Only 600 images were able to be downloaded due to the slow speed of the downloader. 600 images took about a day to download.

Network Design

The neural network was relatively deep (25 layers) due to the complexity of the learning task. Below is the outline of the layers:

- 1 conv. layer, 3 channel input, 64 channel output, and max pooling
- 1 conv. layer, 64 channel input, 64 channel output, and max pooling
- 16 conv. layers, 64 channel input, 64 channel output, skip connections, and batch normalization
- 1 conv. layer, 64 channel input, 128 channel output, skip connections, and batch normalization
- 4 conv. layers, 128 channel input, 128 channel output, skip connections, and batch normalization
- 2 linear layers

YOLO Vector Assignment

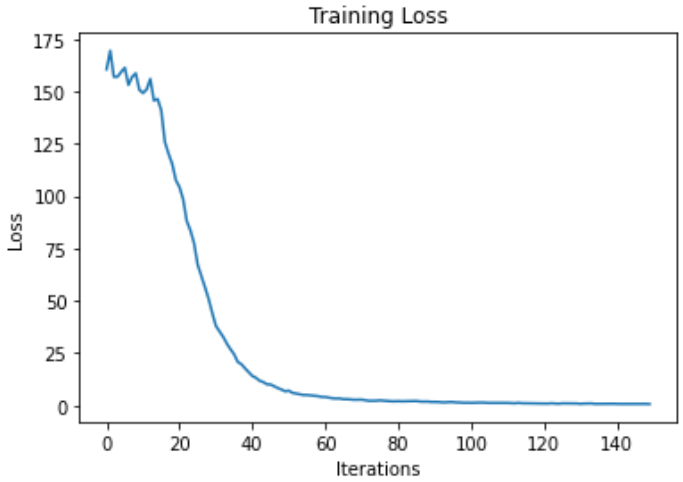
Assignment of the YOLO vector for each image worked rather similarly to RegionProposalGenerator. The 128x128 pixel images were split up into 36 cells. Each cell was 20x20 pixels and the outer border of the image was ignored. Each cell contained 5 anchor boxes with varying aspect ratios, and each object in the image was assigned to an appropriate anchor box.

The YOLO vector served as the “ground truth” and MSE (mean squared error) was used as a loss function.

At first, the network would often predict bboxs with either 0 width or 0 height, which would result in 0 IoU to the ground truth. In order to drive the network to predict non-zero, bh and bw of the YOLO vector are multiplied by 100 prior to computing the loss.

Training

Training was run for 150 epochs, with a learning rate of 0.5e-4, momentum of 0.9, and batch size of 5. The plot below shows the loss decreasing with each training epoch.



Validation

During inference, an object is predicted to be present if the first element of the vector [obj_present] is greater than 0.05. The predicted bbox is then compared to all of the ground truth bboxs.

If there is an overlap (IoU > 0), then that predicted label and bbox are assigned to that object.

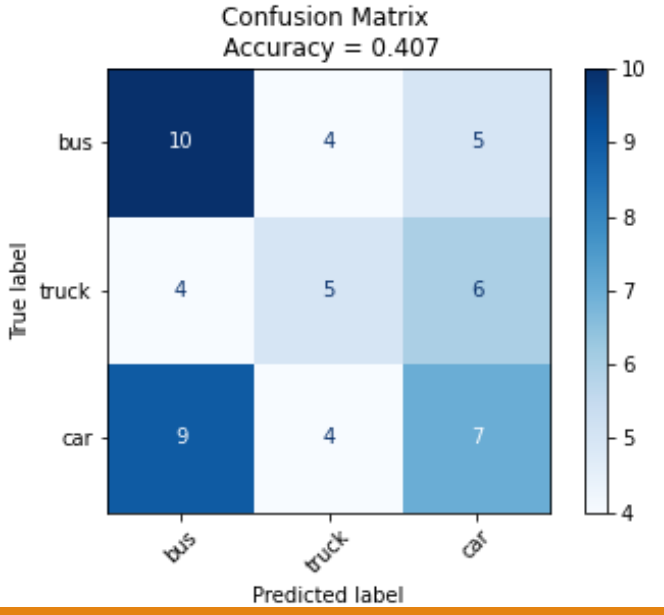
If there is no overlap, then that is counted as a false alarm

Results

False alarm rate: 60.58%

Average Intersection Over Union: 0.093

Class label accuracy: 40.7%
(not counting false alarms)



Results (cont.)

Here are some sample images. Ground truth bboxs are outlined in white while predicted bboxs are outlined in green. Note there seemed to be an issue scaling the ground truth bboxs.