ECE 60146 Deep Learning
Homework 6
03/22/2023

Name: *Nikita Ravi (ravi30)*

1 Introduction

The objective of this homework assignment was to be able to apply the YOLO network and skip connections for the purpose of multi-object detection and localization using Microsoft's COCO Dataset.

2 Methodology

2.1 COCO Downloader

The methodology used to download the images and its respective labels and bounding boxes from the COCO dataset is as follows

1. Get all the image ids that fit all the permutations of the categories (bus, cat, pizza)
2. Iterate through the list of image ids and for each image id create a dictionary containing information on the image id, path to image, and the bounding boxes
3. If the number of annotations are associated to the three categories we care about and the area of the annotation bounding box is greater than 4096, then we add the necessary values to the dictionary created in the previous step
4. The dictionary for each image is then appended to a pandas DataFrame for easy access

The number of training images obtained from the COCO dataset is 6883 and the number of validation images obtained are 3491

2.1.1 Implementation

```
# %%
from pycocotools.coco import COCO
from PIL import Image
import numpy as np
import os
import pandas as pd
import torchvision.transforms as tvt
from torchvision.io import read_image
import argparse
from tqdm import tqdm
import itertools
from pprint import pprint

# %%
```

```
# Global Variables
train_directory = r"/scratch/gilbreth/dfarache/ece60146/Nikita/hw06/train2014"
val_directory = r"/scratch/gilbreth/dfarache/ece60146/Nikita/hw06/val2014"
train_annotations = r"/scratch/gilbreth/dfarache/ece60146/Nikita/hw06/annotations/instances_train2014.json"
val_annotations = r"/scratch/gilbreth/dfarache/ece60146/Nikita/hw06/annotations/instances_val2014.json"


categories = ["bus", "cat", "pizza"]
catIds = [6, 17, 59]
categories = {"bus": 6, "cat": 17, "pizza": 59}
inverse_categories = {6: "bus", 17: "cat", 59: "pizza"}


columns = ["id", "path_to_image", "bus", "cat", "pizza"]
new_image_size = 256
resize_image = tvt.Compose([tvt.Resize((new_image_size,new_image_size))])


train = False
coco = COCO(train_annotations) if train else COCO(val_annotations)
path_to_dir = "/scratch/gilbreth/dfarache/ece60146/Nikita/hw06/Train" if train else "/scratch/gilbreth/dfarache/ece60146/Nikita/hw06/Val"

# %%
def get_all_image_ids():
    all_image_ids = []

    # Get all combinations of the categories
    for idx in range(1, len(categories) + 1):
        combination = [list(i) for i in itertools.combinations(categories, idx)]

        for class_labels in combination:
            category_ids = coco.getCatIds(catNms=class_labels)
            image_ids = coco.getImgIds(catIds=category_ids)

            for image_id in image_ids:
                if(image_id not in all_image_ids):
                    all_image_ids.append(image_id)

    print(f"Total number of images for all the combinations of the categories: {len(all_image_ids)}")
    return all_image_ids

all_image_ids = get_all_image_ids()

# %%
def resize_bbox(bbox, original_width, original_height):
    # Resize bbox
    annotation_width, annotation_height = bbox[2], bbox[3]
    annotation_left_x, annotation_left_y = bbox[0], bbox[1]
```

```
63
64      x_scale = new_image_size / original_width
65      y_scale = new_image_size / original_height
66
67      new_bbox_width = x_scale * annotation_width
68      new_bbox_height = y_scale * annotation_height
69      new_x1 = x_scale * annotation_left_x
70      new_y1 = y_scale * annotation_left_y
71
72      new_x2 = new_x1 + new_bbox_width
73      new_y2 = new_y1 + new_bbox_height
74
75      # Add everything to the list to add to the df
76      ann_bbox = [new_x1, new_y1, new_x2, new_y2]
77      return ann_bbox
78
79  # %%
80  def resize_image_bbox(image, path_to_image, Bboxs):
81      directory = "/scratch/gilbreth/dfarache/ece60146/Nikita/hw06/train2014
        " if train else "/scratch/gilbreth/dfarache/ece60146/Nikita/hw06/
        val2014"
82      new_bboxs = {"bus": [], "cat": [], "pizza": []}
83
84      try:
85          image_filename = image["file_name"]
86          original_width, original_height = image["width"], image["height"]
87
88          original_image = resize_image(read_image(os.path.join(directory,
        image_filename)))
89          original_image = original_image.repeat(3,1,1) if original_image.
        size()[0] == 1 else original_image # Check if image has three channels
90          original_image = tvt.functional.to_pil_image(original_image).
        convert("RGB") # convert to RGB image
91
92          for label, bboxs in Bboxs.items():
93              for idx in range(len(bboxs)):
94                  if(bboxs[idx] != -1):
95                      resized_bbox = resize_bbox(bboxs[idx], original_width,
         original_height)
96
97                      if(label == "bus"):
98                          new_bboxs["bus"].append(resized_bbox)
99                          new_bboxs["cat"].append(-1)
100                         new_bboxs["pizza"].append(-1)
101
102                     elif(label == "cat"):
103                         new_bboxs["cat"].append(resized_bbox)
104                         new_bboxs["bus"].append(-1)
105                         new_bboxs["pizza"].append(-1)
106
107                     elif(label == "pizza"):
108                         new_bboxs["pizza"].append(resized_bbox)
109                         new_bboxs["cat"].append(-1)
110                         new_bboxs["bus"].append(-1)
```

```
111
112         original_image.save(path_to_image)
113         return new_bboxs
114
115     except Exception as e:
116         print(f"Exception: {e}")
117         return False
118
119 # %%
120 dataset = {}
121 for image_id in tqdm(all_image_ids):
122     image_info = {"Id": None,
123                   "Path to Image": None,
124                   "Bbox": {"bus": [], "cat": [], "pizza": []}}
125
126     image = coco.loadImgs(ids=image_id)[0]
127     annotation_ids = coco.getAnnIds(imgIds=image["id"], iscrowd=False)
128     annotations = coco.loadAnns(annotation_ids)
129
130     for annotation in annotations:
131         if(annotation["category_id"] in catIds):
132             if(annotation["area"] > (64 * 64)):
133                 path_to_image = os.path.join(path_to_dir, image["file_name
    "])
134                 image_info["Id"] = image_id
135                 image_info["Path to Image"] = path_to_image
136
137                 label_name = inverse_categories[annotation['category_id']]
138                 if(label_name == "bus"):
139                     image_info['Bbox']["bus"].append(annotation['bbox'])
140                     image_info['Bbox']["cat"].append(-1)
141                     image_info['Bbox']["pizza"].append(-1)
142                 elif(label_name == "cat"):
143                     image_info['Bbox']["cat"].append(annotation['bbox'])
144                     image_info['Bbox']["bus"].append(-1)
145                     image_info['Bbox']["pizza"].append(-1)
146                 elif(label_name == "pizza"):
147                     image_info['Bbox']["pizza"].append(annotation['bbox'])
148                     image_info['Bbox']["cat"].append(-1)
149                     image_info['Bbox']["bus"].append(-1)
150
151         if(image_info["Path to Image"] != None):
152             resized_bboxs = resize_image_bbox(image, image_info["Path to
    Image"], image_info["Bbox"])
153             if(resized_bboxs):
154                 image_info["Bbox"] = resized_bboxs
155                 dataset[image["file_name"]] = image_info
156
157 print(f'Total Images downloaded {len(dataset)}')
158
159 # %%
160 def create_dataframe(dataset):
161     df_filename = "train_data.csv" if train else "test_data.csv"
162     df = pd.DataFrame(columns=columns)
```

```
    ids, paths_to_image, bus_bboxs, cat_bboxs, pizza_bboxs = [], [], [], [], []
    for image_filename, image_info in dataset.items():
        # image_info = {id:str, path_to_image:str, bboxs:dict}
        ids.append(image_info["Id"])
        paths_to_image.append(image_info["Path to Image"])
        bus_bboxs.append(image_info["Bbox"]["bus"])
        cat_bboxs.append(image_info["Bbox"]["cat"])
        pizza_bboxs.append(image_info["Bbox"]["pizza"])

    assert len(bus_bboxs) == len(cat_bboxs) == len(pizza_bboxs) == len(ids) == len(paths_to_image), f"Lengths of data not matched: {len(bus_bboxs)}, {len(cat_bboxs)}, {len(pizza_bboxs)}, {len(ids)}, {len(paths_to_image)}"
    df["id"] = ids
    df["path_to_image"] = paths_to_image
    df["bus"] = list(bus_bboxs)
    df["cat"] = list(cat_bboxs)
    df["pizza"] = list(pizza_bboxs)
    df.to_csv(df_filename)

create_dataframe(dataset)

# %%
```

Listing 1: COCO Downloader

2.1.2 Inputs Sample

The following image is a sample of the training data used for the YOLO network

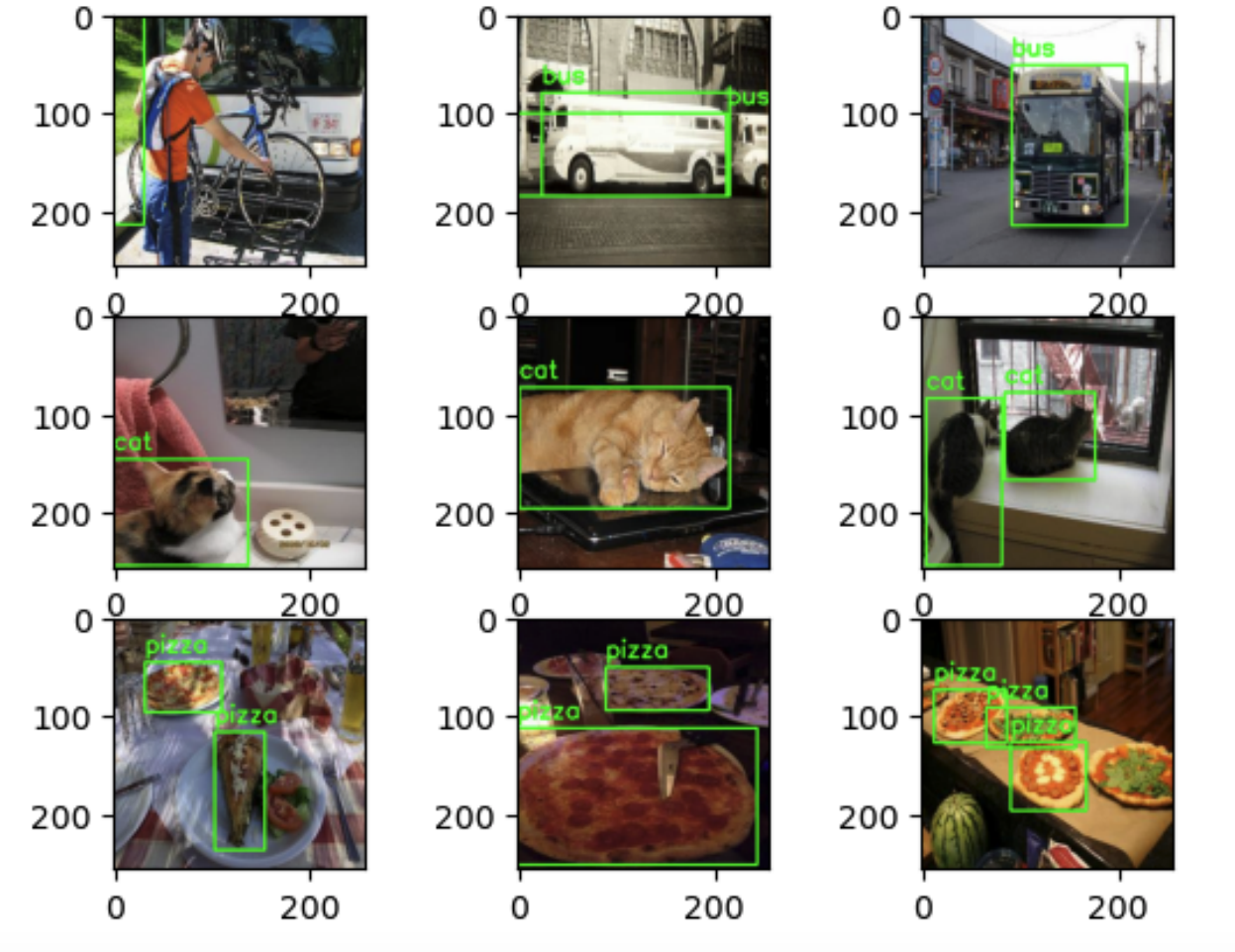

Figure 1: Inputs

2.2 Dataloader

2.2.1 Creation of the YOLO Tensor

The YOLO tensor is created through the following steps

1. Iterate through all the foreground objects present in the image downloaded from the COCO dataset
2. Overlay the image with a 6×6 grid where each cell has a dimension of 42×42 pixels
3. Compute the coordinates for the center of the object in the image

4. Obtain the cell coordinate that corresponds to the object found in the image
5. Assign to a variable, the height and width of the bounding box
6. Calculate the x-coordinate displacement δx and the y-coordinate displacement δy from the center of the bounding box and the center of the cell
7. Calculate the aspect ratio (bounding box height divided by the bounding box width). For this assignment, the aspect ratios used are 1/5, 1/3, 1/1, 3/1, and 5/1
8. Create a vector, called the yolo vector, of size 8 to store the indicator of an object presence, δx, δy, the height of the bounding box, the width of the bounding box, and the one hot encoding of the class label
9. Append the yolo vector to a yolo tensor by the cell index and anchor box index
10. Finally create an augmented version of the yolo tensor to throw all the probability mass into an extra ninth element of the yolo vector if no object is present

```
def create_yolo_tensor(bboxs, labels, num_images_in_batch=1):
    # Inspired by Professor Kak's run_code_for_training_multi_instance_detection function
    height_center_bb = torch.zeros(num_images_in_batch, 1).float()
    width_center_bb = torch.zeros(num_images_in_batch, 1).float()
    object_bb_height = torch.zeros(num_images_in_batch, 1).float()
    object_bb_width = torch.zeros(num_images_in_batch, 1).float()
    yolo_tensor = torch.zeros(num_yolo_cells, num_anchor_boxes, yolo_vector_size)

    # jdx represents the index of the object in the foreground of the image
    for jdx in range(max_num_objects):
        object_center_y_coord = (bboxs[jdx, 1].float() + bboxs[jdx, 3].float()) / 2.0 # Just check the batch and iterate through each bbox for y1 and y2
        height_center_bb = object_center_y_coord.int()

        object_center_x_coord = (bboxs[jdx, 0].float() + bboxs[jdx, 2].float()) / 2.0 # Just check the batch and iterate through each bbox for x1 and x2
        width_center_bb = object_center_x_coord.int()

        object_bb_height = bboxs[jdx, 3] - bboxs[jdx, 1] # Height of the bounding box
        object_bb_width = bboxs[jdx, 2] - bboxs[jdx, 0] # Width of the bounding box

        if(object_bb_height < 4.0 or object_bb_width < 4.0):
            # Disregard bounding boxes that have a height or width of less than an arbitrary number 4
            continue

```

```
        # Get the cell row and column index that corresponds to the center of the bounding box
        cell_row_idx = torch.clamp((height_center_bb / yolo_interval).int(), max=num_cells_image_height - 1)
        cell_col_idx = torch.clamp((width_center_bb / yolo_interval).int(), max=num_cells_image_width - 1)

        ### Get the height of the bounding box divided by the actual height of the cell
        bh = object_bb_height.float() / yolo_interval
        bw = object_bb_width.float() / yolo_interval

        ### Calculate del_x and del_y
        # del_x is the x_coordinate displacement from the center of the bbox and the center of the cell
        # del_y is the y_coordinate displacement from the center of the bbox and the center of the cell

        # Calculate the center of the cell (i,j) coordinates
        cell_center_i = cell_row_idx * yolo_interval + float(yolo_interval) / 2.0
        cell_center_j = cell_col_idx * yolo_interval + float(yolo_interval) / 2.0

        # Compute del_x and del_y
        del_x = (object_center_x_coord.float() - cell_center_j.float()) / yolo_interval
        del_y = (object_center_y_coord.float() - cell_center_i.float()) / yolo_interval

        ### Get the class label
        class_label_of_object = int(labels[jdx].item())
        if(class_label_of_object == 13):
            # Disregard labels with class label 13
            continue

        ### Get Aspect Ratio
        aspect_ratio = object_bb_height.float() / object_bb_width.float()
        anchor_box_idx = 0
        if aspect_ratio <= 0.2:                 anchor_box_idx = 0
                                                ## (45)
        if 0.2 < aspect_ratio <= 0.5:           anchor_box_idx = 1
                                                ## (46)
        if 0.5 < aspect_ratio <= 1.5:           anchor_box_idx = 2
                                                ## (47)
        if 1.5 < aspect_ratio <= 4.0:           anchor_box_idx = 3
                                                ## (48)
        if aspect_ratio > 4.0:                  anchor_box_idx = 4

        ### Create the yolo vector
        yolo_vector = torch.FloatTensor([0, del_x.item(), del_y.item(), bh.item(), bw.item(), 0, 0, 0])
        yolo_vector[0] = 1 # Object exists
        yolo_vector[5 + class_label_of_object] = 1
```

```
63
64          ### Assign to yolo tensor
65          yolo_cell_index = cell_row_idx.item() * num_cells_image_width +
    cell_col_idx.item()
66          yolo_tensor[yolo_cell_index, anchor_box_idx] = yolo_vector  # 1
    x36x5x8
67
68      ## Create an augmented yolo tensor where if no object is present,
    throw all the prob mass into the extra 9th element of yolo vector
69      yolo_tensor_aug = torch.zeros(num_yolo_cells, num_anchor_boxes,
    yolo_vector_size + 1).float()
70      yolo_tensor_aug[:,:,:-1] = yolo_tensor
71
72      # If no object is present, throw all the prob mass into the extra 9th
    element of the yolo vector
73      for icx in range(num_yolo_cells):
74          for iax in range(num_anchor_boxes):
75              if(yolo_tensor_aug[icx, iax, 0] == 0):
76                  yolo_tensor_aug[icx, iax, -1] = 1
77
78      return yolo_tensor_aug
```

Listing 2: Creation of YOLO Tensor

2.2.2 The PyTorch Dataset Class

The dataset class returns the image tensor, the bounding boxes in that image tensor, the labels in that image tensor, and finally its corresponding yolo tensor for the dataloaders

```
1  class GenerateDataset(torch.utils.data.Dataset):
2      def __init__(self, df, transform=None):
3          super().__init__()
4          self.df = df
5          self.transform = transform
6
7      def __return_integer_encoding(self, category):
8          categories = {"bus": 0, "cat": 1, "pizza": 2}
9          return int(categories[category])
10
11     def __convert_string_to_list(self, info):
12         try:
13             return json.loads(info)
14         except Exception as e:
15             print(info)
16
17     def __return_image(self, info):
18         path_to_image = os.path.join(r"/scratch/gilbreth/dfarache/ece60146
    /Nikita/hw06", info["path_to_image"])
19         image = Image.open(path_to_image)
20         image = self.transform(image) if self.transform else image
21
22         return image
23
24     def __return_bbox_and_labels(self, info):
```

```
        bus_bbox = self.__convert_string_to_list(info["bus"])
        cat_bbox = self.__convert_string_to_list(info["cat"])
        pizza_bbox = self.__convert_string_to_list(info["pizza"])
        label_bbox_dict = {"bus": bus_bbox, "cat": cat_bbox, "pizza": pizza_bbox}

        assert len(bus_bbox) == len(cat_bbox) == len(pizza_bbox), f"Number of annotations are not the same {len(bus_bbox)} != {len(cat_bbox)} != {len(pizza_bbox)}"

        labels = torch.zeros(max_num_objects, dtype=torch.uint8) + 13 # 13 was randomly selected to differentiate from labelled objects in the foreground
        bboxs = torch.zeros(max_num_objects, 4, dtype=torch.uint8)

        row = 0
        for label, bbs in label_bbox_dict.items():
            for jdx in range(len(bbs)):
                bb = bbs[jdx]
                if(bb != -1 and row < max_num_objects):
                    bboxs[row] = torch.tensor(bb, dtype=torch.float)
                    labels[row] = self.__return_integer_encoding(label)
                    if(row < max_num_objects):
                        row += 1

        return labels, bboxs

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        image_info = self.df.iloc[idx]
        image = self.__return_image(image_info)
        labels, bboxs = self.__return_bbox_and_labels(image_info)
        yolo_tensor = create_yolo_tensor(bboxs, labels)

        return image, bboxs, labels, yolo_tensor
```

Listing 3: Generate Dataset

```
def get_dataloader(path, debug=False):
    # Constants
    transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

    # Create PyTorch Datasets and Dataloader
    df = pd.read_csv(path)
    dataset = GenerateDataset(df, transform)

    if(debug):
        print(f"Length of dataset: {len(dataset)}")
        print(f"Image size: {dataset[0][0].shape}")
        print("Bounding Boxes")
        print(dataset[0][1])
```

```
        print("Labels")
        print(dataset[0][2])
        print("Yolo Tensor")
        print(dataset[0][3])

    dataloader = torch.utils.data.DataLoader(dataset, batch_size=
    batch_size, num_workers=2, shuffle=True, drop_last=True)
    return dataloader

# Get Trainloader
trainloader = get_dataloader(path=r"/scratch/gilbreth/dfarache/ece60146/
    Nikita/hw06/train_data.csv", debug=True)
testloader = get_dataloader(path=r"/scratch/gilbreth/dfarache/ece60146/
    Nikita/hw06/test_data.csv", debug=True)
```

Listing 4: Skip Connections

2.3 YOLO Algorithm

2.3.1 The Networks

The network designed is similar to the one used in homework 5 - single object dectection and localization with a slight modification. The returned value from the network is going to be the predicted yolo vector which has a final of 1620 output features. The entire network has 80 layers and 55,140,564 learnable parameters.

```
class SkipBlock(nn.Module):
    # Inspired by Professor Kak's SkipBlock class
    def __init__(self, in_ch, out_ch, downsample=False, skip_connections=
    True):
        super(SkipBlock, self).__init__()
        self.downsample = downsample
        self.skip_connections = skip_connections
        self.in_ch = in_ch
        self.out_ch = out_ch
        self.conv1 = nn.Conv2d(self.in_ch, self.out_ch, kernel_size=3,
    stride=1, padding=1)
        self.conv2 = nn.Conv2d(self.in_ch, self.out_ch, kernel_size=3,
    stride=1, padding=1)
        self.bn = nn.BatchNorm2d(self.out_ch)

        if(downsample):
            self.downsampler = nn.Conv2d(self.in_ch, self.out_ch,
    kernel_size=1, stride=2)

    def forward(self, x):
        identity = x
        out = self.conv1(x)
        out = self.bn(out)
        out = F.relu(out)
        if(self.in_ch == self.out_ch):
            out = self.conv2(out)
            out = self.bn(out)
```

```
            out = F.relu(out)
        if(self.downsample):
            out = self.downsampler(out)
            identity = self.downsampler(identity)
        if(self.skip_connections):
            if(self.in_ch == self.out_ch):
                out = out + identity
            else:
                out = torch.cat((out[:,:self.in_ch,:,:] + identity, out[:,
    self.in_ch:, :, :] + identity), dim=1)
        return out
```

Listing 5: Skip Connections

```
class NetForYolo(nn.Module):
    # Inspired by Professor Kak's NetForYolo class
    def __init__(self, skip_connections=True, depth=8):
        super(NetForYolo, self).__init__()
        self.skip_connections = skip_connections
        self.depth = depth // 2
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size
    =3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=64, out_channels=64,
    kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.bn1 = nn.BatchNorm2d(num_features=64)
        self.bn2 = nn.BatchNorm2d(num_features=128)
        # self.bn3 = nn.BatchNorm2d(num_features=256)
        self.skip64_arr = nn.ModuleList()
        for idx in range(self.depth):
            self.skip64_arr.append(SkipBlock(in_ch=64, out_ch=64,
    skip_connections=self.skip_connections))

        self.skip64ds = SkipBlock(in_ch=64, out_ch=64, downsample=True,
    skip_connections=self.skip_connections)
        self.skip64to128 = SkipBlock(in_ch=64, out_ch=128,
    skip_connections=self.skip_connections)
        self.skip128_arr = nn.ModuleList()
        for idx in range(self.depth):
            self.skip128_arr.append(SkipBlock(in_ch=128, out_ch=128,
    skip_connections=self.skip_connections))

        self.skip128ds = SkipBlock(in_ch=128, out_ch=128, downsample=True,
     skip_connections=self.skip_connections)
        # self.skip128to256 = SkipBlock(in_ch=128, out_ch=256,
    skip_connections=self.skip_connections)
        # self.skip256_arr = nn.ModuleList()
        # for idx in range(self.depth):
        #     self.skip256_arr.append(SkipBlock(in_ch=256, out_ch=256,
    skip_connections=self.skip_connections))

        # self.skip256ds = SkipBlock(in_ch=256, out_ch=256, downsample=
    True, skip_connections=self.skip_connections)
        self.fc_seqn = nn.Sequential(
```

```
            nn.Linear(in_features=128*16*16, out_features=6*6*5*9),
#             nn.ReLU(inplace=True),
#             nn.Linear(in_features=4096, out_features=2048),
#             nn.ReLU(inplace=True),
#             nn.Linear(in_features=2048, out_features=6*6*5*9) # 6x6 grid overlaying the image, 5 anchor boxes, length 9 yolo vector
        )

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        for idx, skip64 in enumerate(self.skip64_arr[:self.depth//4]):
            x = skip64(x)
        x = self.skip64ds(x)
        for idx, skip64 in enumerate(self.skip64_arr[:self.depth//4]):
            x = skip64(x)
        x = self.bn1(x)
        x = self.skip64to128(x)
        for idx, skip128 in enumerate(self.skip128_arr[:self.depth//4]):
            x = skip128(x)
        x = self.bn2(x)
        x = self.skip128ds(x)
        x = x.view(-1, 128*16*16)
        x = self.fc_seqn(x)
        return x
```

Listing 6: YOLO Network

2.3.2 Training Algorithm

The training algorithm consists of passing the image into the network shown above to obtain the predicted yolo tensor. We then compute the binary cross entropy loss for the object presence indicator, MSE loss for the regression elements of the yolo vectors in the tensor, and finally cross entropy loss on the predicted class labels. The training function accommodates for variable batch sizes through indexing of the yolo tensors and only checks yolo vectors that have an object.

```
def train(net, trainloader):
    # Inspired by Professor Kak's run_code_for_training_multi_instance_detection function
    net = net.to(device)
    presence_average_loss = []
    regression_average_loss = []
    class_average_loss = []

    print("Training started")

    for epoch in range(1, epochs + 1):
        print(f"Epoch: {epoch} / {epochs}")
        running_bce_loss = 0.0
        running_mse_loss = 0.0
        running_ent_loss = 0.0

```

```
16        for batch_idx, (images_in_batch, _, _, yolo_tensors_in_batch) in
  enumerate(trainloader):
17            images_in_batch = images_in_batch.to(device)
18            yolo_tensors_in_batch = yolo_tensors_in_batch.to(device)
19
20            optimizer.zero_grad()
21            output = net(images_in_batch)
22            # print(output.shape)
23            output = output.view(batch_size, num_yolo_cells,
  num_anchor_boxes, yolo_vector_size+1)
24
25            total_bce_loss = torch.tensor(0.0, requires_grad=True).float()
  .to(device)
26            total_mse_loss = torch.tensor(0.0, requires_grad=True).float()
  .to(device)
27            total_ent_loss = torch.tensor(0.0, requires_grad=True).float()
  .to(device)
28
29            # Get Yolo Vectors where object presence is not zero
30            sigmoid = nn.Sigmoid()
31            object_presence = torch.nonzero(output[:, :, :, 0])
32            bce_loss = criterion1(sigmoid(output[:, :, :, 0]),
  yolo_tensors_in_batch[:, :, :, 0])
33            total_bce_loss += bce_loss
34
35            # Compute Regression Loss
36            predicted_regression_vector = output[object_presence[:,0],
  object_presence[:,1], object_presence[:,2], 1:5]
37            target_regression_vector = yolo_tensors_in_batch[
  object_presence[:,0], object_presence[:,1], object_presence[:,2], 1:5]
38            mse_loss = criterion2(predicted_regression_vector,
  target_regression_vector)
39            total_mse_loss += mse_loss
40
41            # Compute Cross Entropy Loss
42            class_probs_vector = output[object_presence[:,0],
  object_presence[:,1], object_presence[:,2], 5:]
43            target_class_vector = torch.argmax(yolo_tensors_in_batch[
  object_presence[:,0], object_presence[:,1], object_presence[:,2], 5:],
  dim=1)
44            cross_loss = criterion3(class_probs_vector,
  target_class_vector)
45            total_ent_loss += cross_loss
46
47            total_ent_loss.backward(retain_graph=True)
48            total_mse_loss.backward(retain_graph=True)
49            total_bce_loss.backward(retain_graph=True)
50
51            optimizer.step()
52
53            running_bce_loss += total_bce_loss.item()
54            running_mse_loss += total_mse_loss.item()
55            running_ent_loss += total_ent_loss.item()
56
```

```
57
58              if(batch_idx % 20 == 19):
59                  print(f"[Epoch: {epoch}/{epochs}, Batch Idx= {batch_idx +
    1}] ------- Average BCE Loss: {running_bce_loss / float(20)}, Average
    MSE Loss: {running_mse_loss / float(20)}, Average Cross Entropy Loss: {
    running_ent_loss / float(20)}")
60
61                  presence_average_loss.append(running_bce_loss / float(20))
62                  regression_average_loss.append(running_mse_loss / float
    (20))
63                  class_average_loss.append(running_ent_loss / float(20))
64
65                  torch.save(net.state_dict(), path_to_model)
66
67                  running_bce_loss = 0.0
68                  running_mse_loss = 0.0
69                  running_ent_loss = 0.0
70
71      return presence_average_loss, regression_average_loss,
    class_average_loss
72
73 presence_loss, regression_loss, class_loss = train(net, trainloader)
```

Listing 7: Training

The parameters to the training algorithm were

Parameters	Value
Epochs	12
Learning Rate	1e−6
Optimizer	Adam
Betas	(0.9, 0.99)
Criterion 1	Binary Cross Entropy
Criterion 2	Mean Squared Error
Criterion 3	Cross Entropy

The following image illustrates the binary cross entropy loss, the regression loss, and the cross entropy loss across all 12 epochs

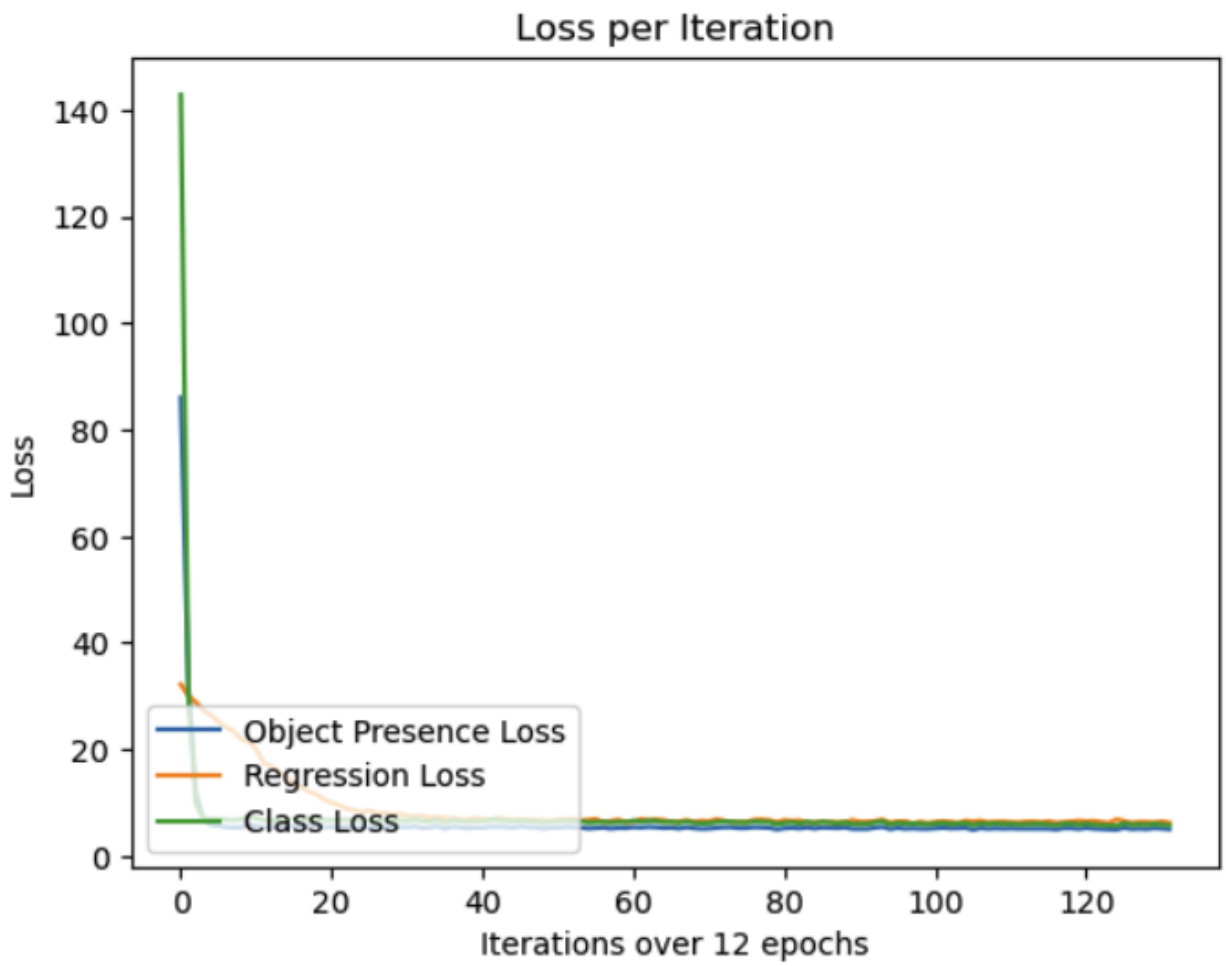

Figure 2: Training Losses

2.3.3 Testing Algorithm

In the testing function, we iterate across one batch, across one batch axis to obtain the predicted bounding boxes and labels for each image. A comparision of the ground truth and the predictions are illustrated in figure **??**. The implementation for the testing function is shown below.

```
def test(net, data):
    net.load_state_dict(torch.load(path_to_model))
    net.eval()
    net.to(device)

    image, bboxs_in_image, labels_in_image, yolo_tensor = data

    # Add an extra dimension to the image to pass it to the network
    image_for_network = torch.unsqueeze(deepcopy(image), dim=0).to(device)

```

```
    predicted_yolo_tensor = net(image_for_network)
    predicted_yolo_tensor = predicted_yolo_tensor.view(1, num_yolo_cells,
num_anchor_boxes, yolo_vector_size+1)
    yolo_tensor = torch.unsqueeze(yolo_tensor, dim=0)

    predictions = {}
    softMax = nn.Softmax(dim=0)

    for icx in range(num_yolo_cells):
        for iax in range(num_anchor_boxes):
            predicted_yolo_vector = predicted_yolo_tensor[0, icx, iax]
            target_label = torch.argmax(softMax(predicted_yolo_vector[5:])
)

            if target_label.item() != 3:
                predictions[(icx, iax)] = predicted_yolo_vector

    # Display Image with annotations
    image = np.asarray(tvt.ToPILImage()(image / 2  + 0.5))

    for key, yolo_vector in predictions.items():
        cell_idx, anchor_box_idx = key
        object_presence, del_x, del_y, bh, bw, bus, cat, pizza, extra =
yolo_vector

        predicted_label = torch.argmax(softMax(torch.tensor([bus, cat,
pizza, extra])))

        if predicted_label == 0: label = 'bus'
        if predicted_label == 1: label = 'cat'
        if predicted_label == 2: label = 'pizza'

        cell_col_idx = cell_idx % num_cells_image_width
        cell_row_idx = cell_idx // num_cells_image_height

        cell_center_i = cell_row_idx * yolo_interval + float(yolo_interval
) / 2.0
        cell_center_j = cell_col_idx * yolo_interval + float(yolo_interval
) / 2.0

        obj_center_x = del_x * yolo_interval + cell_center_j
        obj_center_y = del_y * yolo_interval + cell_center_i

        bh *= yolo_interval
        bw *= yolo_interval

        x1 = int(obj_center_x - bw / 2)
        y1 = int(obj_center_y - bh / 2)

        x2 = int(obj_center_x + bw / 2)
        y2 = int(obj_center_y + bh / 2)

        image = cv2.rectangle(image,  (x1, y1), (x2, y2), color=(0, 0,
255), thickness=1)
```

```
58          image = cv2.putText(image, label, (x1, y1-10), cv2.
    FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
59
60      # Ground Truth
61      for idx in range(len(bboxs_in_image)):
62          x1 = int(bboxs_in_image[idx,0])
63          y1 = int(bboxs_in_image[idx,1])
64          x2 = int(bboxs_in_image[idx,2])
65          y2 = int(bboxs_in_image[idx,3])
66
67          gt_label = int(labels_in_image[idx])
68          if(gt_label == 13):
69              continue
70          gt_label = inverse_categories[gt_label]
71
72          image = cv2.rectangle(image,  (x1, y1), (x2, y2), color=(0, 255,
    0), thickness=1)
73          image = cv2.putText(image, gt_label, (x1, y1-10), cv2.
    FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
74      return image
```

Listing 8: Testing

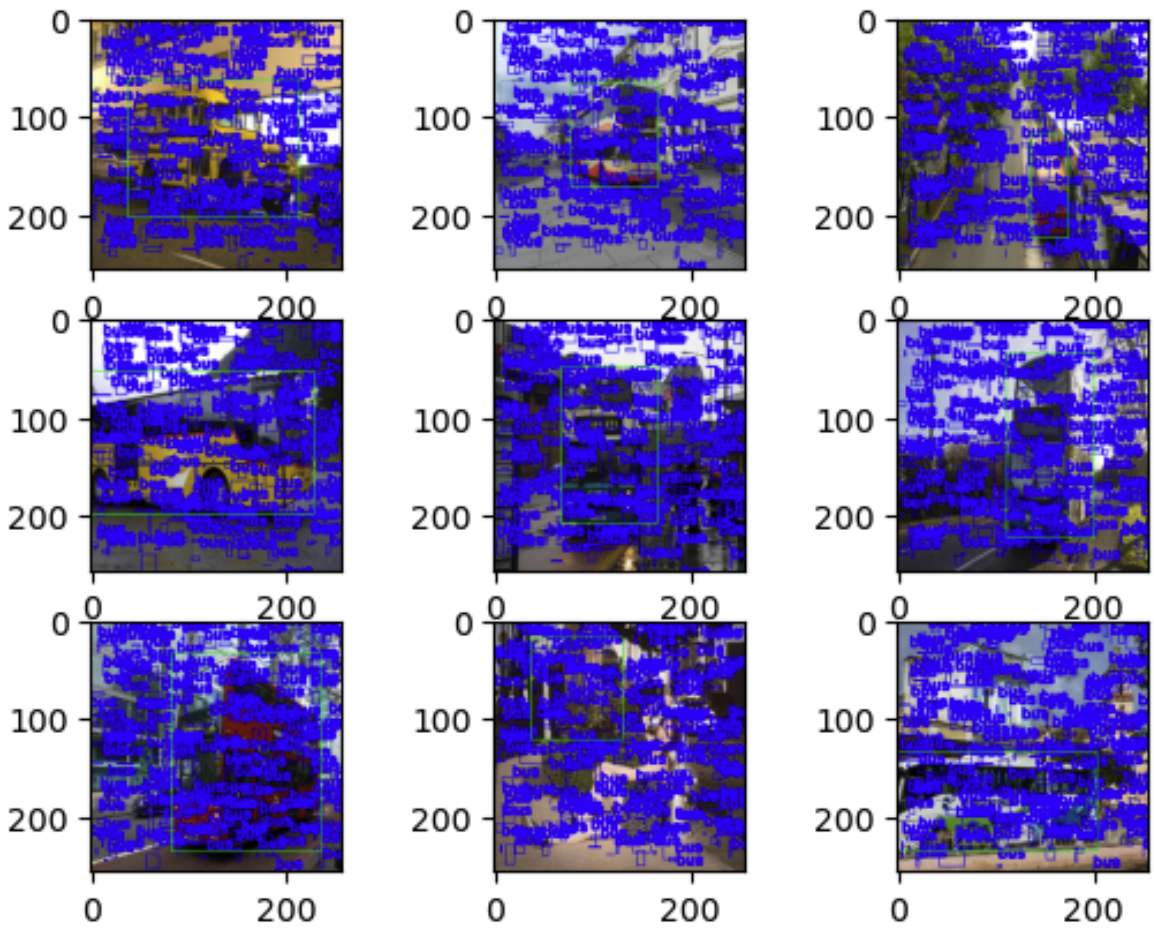

Figure 3: Inputs

3 Evaluation

The YOLO algorithm has a high accuracy rate for identifying the foreground objects in the images. However, with respect to the bounding boxes, the location is usually correct but the size of the bounding box is inaccurate. This could be a problem in the dataset or the way the yolo tensors are created. Maybe removing the augmented yolo tensor could improve the results.