

PURDUE UNIVERSITY
Elmore Family School of Electrical and Computer Engineering
Deep Learning

Homework 6

Adithya Sineesh
Email : asineesh@purdue.edu
Submitted: March 22, 2023

1 Experiment

In this homework, we do the following:

1. Make our own dataset from the MS-COCO dataset for multi-instance object detection.
2. To train a model which can perform multi-instance object detection using the YOLO logic.

I have referred to Prof. Kak's presentation on Multi-instance object detection as well as last year's assignment.

1.1 Dataset Loading

We first download the JSON annotation file for the training and validation set. Then using the COCO API, we load only those images that have at least one object of area 4096 pixels and belonging to the bus, cat or pizza class. This comes to 6883 training and 3491 validation images and they are resized to (256×256) and saved along with the labels and bounding box coordinates for each object (top left and right, width and height of the image) which are modified due to the resizing of the image.

1.2 Model Training

The architecture of the model is similar to the one used in the previous homework and similar to ResNet18. It contains 8 ResBlocks with each of them having two convolutional layers. At the head of the model, there is a fully connected layer that gives a YOLO tensor as its output. The optimizer used was Adam, the learning rate was 0.00001 for the first 80 epochs and 0.000001 for the last 20 epochs.

1.2.1 DataLoading

This involves converting all the bounding box coordinates and labels of an image into YOLO tensor. The following are the steps to obtain it:

1. The image is divided into a number of cells. In this Homework, I have divided the 256×256 image into 64 cells each of shape 32×32 .
2. Each cell has five anchor boxes of ratio $1/5, 1/3, 1, 3, 5$.
3. Then for each object instance in the image, the cell containing the object's center is chosen. The ratio of the height and width of the object helps to choose the anchor box.
4. Then for each object instance in the image, the YOLO vector is constructed as follows:
 - In this homework, the objects are of three classes. Therefore the YOLO vector will be of shape $5 + 3 + 1$. For example, if the object is of the first class, the YOLO vector will look as follows:

| | | | | | | | | |
|---|------------|------------|-----|-----|---|---|---|---|
| 1 | δx | δy | h | w | 1 | 0 | 0 | 0 |
|---|------------|------------|-----|-----|---|---|---|---|

 - The first element of the YOLO vector indicates the presence of an object i.e. 0 for no object and 1 for an object.
 - The next two elements of the YOLO vector, $(\delta x, \delta y)$ are the distance between the center of the bounding box and the center of the cell.
 - The fourth and fifth elements of the YOLO vector (h, w) are the height and width of the bounding box with respect to the cell size.
 - The last four elements of the YOLO vector are reserved for one-hot encoding the object's class. The first three elements correspond to the three classes and the last element is set to one when there is no object.

5. A YOLO tensor in this homework has the shape (64, 5, 9) where 64 is the number of cells, 5 is the number of anchor boxes and 9 is the length of the YOLO vector.
6. After constructing the YOLO vector for all the instances in an image, we insert it into the YOLO tensor in the appropriate index based on the anchor box and cell of the object.
7. For the rest of the cells and anchor boxes the YOLO tensor is filled with the following YOLO vector:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|

8. For this homework, I have implemented this as a method of the Dataset class itself i.e. the `__getitem__()` method returns the image and its YOLO Tensor.

1.2.2 Training

The following are the different losses used:

1. Binary Cross Entropy is used for objectness i.e. on the first element of the YOLO vector.
2. Mean Squared Error is used for bounding box regression i.e. on the next four elements of the YOLO vector.
3. Cross Entropy is used for classification i.e. on the last four elements of the YOLO vector.

During training, the target and predicted YOLO vectors, which are of shape (batch size, num_cells, num_anchbox, 9), are reshaped into (batch size \times num_cells \times num_anchbox, 9). Then we extract those indexes from the first axes where the predicted or the target YOLO vectors indicate the presence of an object i.e. the first element of the YOLO vector is greater than 0.5. We then compute the above losses only on these YOLO vectors.

The three losses are added up and backpropagated through the model. The method described above eliminates the need to loop through the images, cells and anchor boxes in a batch to compute loss.

1.2.3 Evaluation

This involves extracting all bounding boxes and labels for an image from the predicted YOLO tensor. The following are the steps involved in it for a single image:

1. We iterate through all the cells of the YOLO Tensor (taken as 64 for this homework) and save the anchor box for which the object has the highest probability of being present i.e. for the 5 anchor boxes which one has the greatest value in the first element of the YOLO vector. This essentially gives us one YOLO vector for each cell.
2. For all the 64 cells, we then sort them based on the objectness of the YOLO vector corresponding to the best anchor box for the cell.
3. We select the top 5 YOLO vectors (as most of the test images have five or fewer objects) and then proceed to obtain their corresponding bounding box and label if the first element of the YOLO vector is greater than 0.25.
4. The original height and width of the bounding box is obtained by multiplying the cell height and width with the fourth and fifth element of the YOLO vector.
5. The above values along with the cell index where the object center is present and the second and third elements of the YOLO vector help us obtain the top left coordinates of the bounding box.
6. The fifth, sixth and seventh elements of the YOLO vector help us obtain the label of the bounding box.

2 Results

2.1 Dataset

The following are three images from each of the three classes along with the annotation of the objects in them.

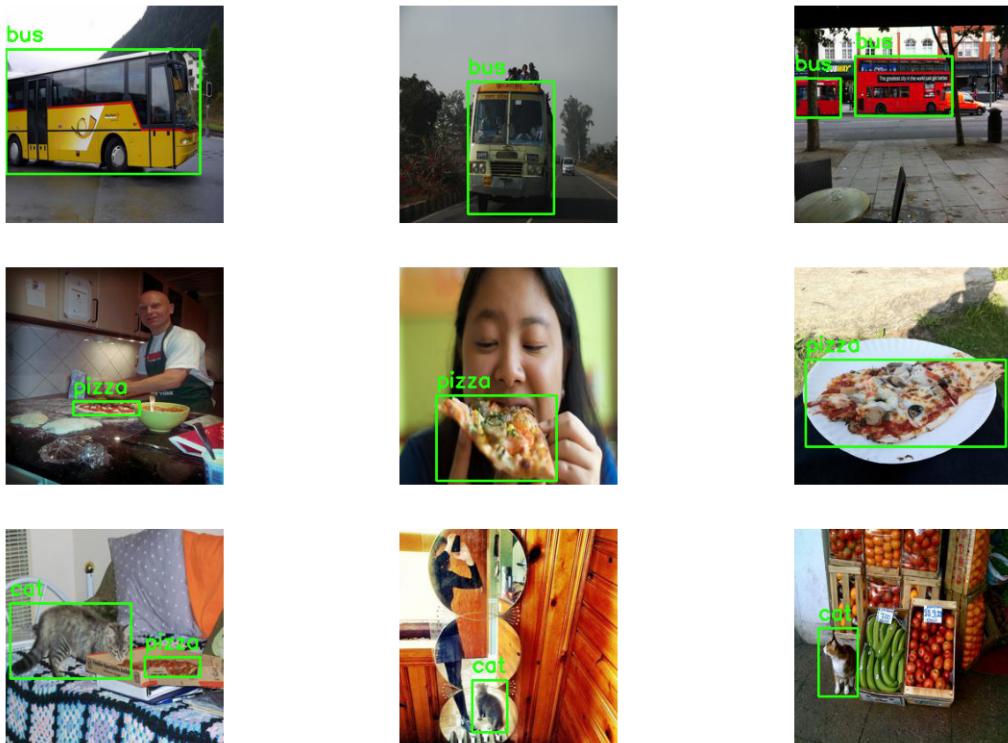


Figure 1: The first row contains images that have at least one bus, the second row contains images that have at least one pizza and the third row contains images that have at least one cat

2.2 Model Training

2.2.1 For Loss

Below, I have plotted the loss vs epochs graph for both the training and testing datasets. The total loss, BCE loss, MSE loss and CE loss are plotted.

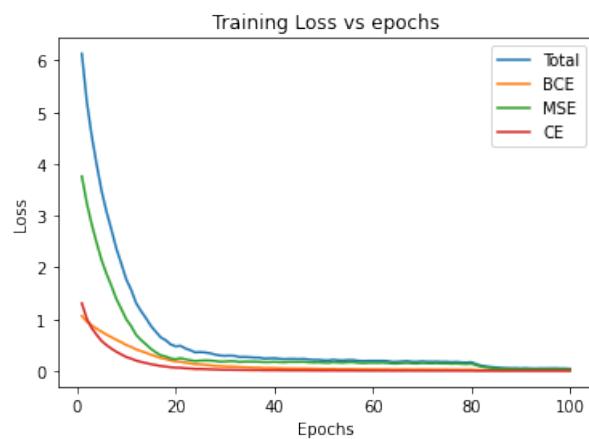


Figure 2: For training

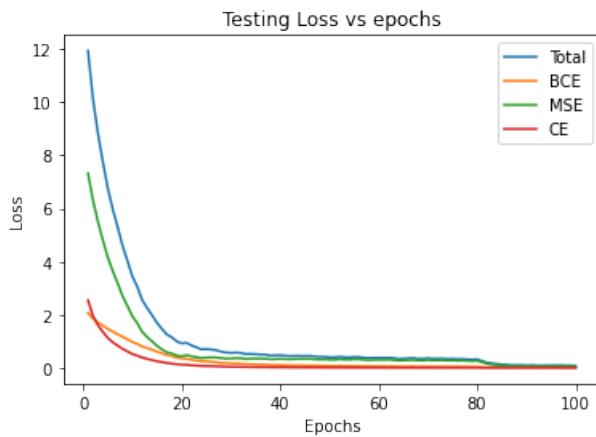
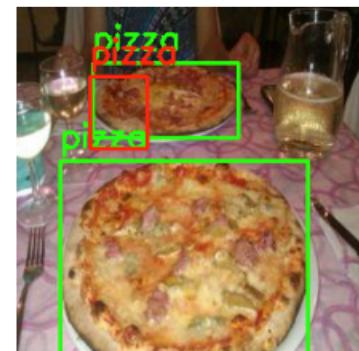
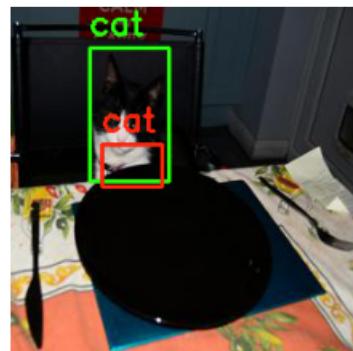
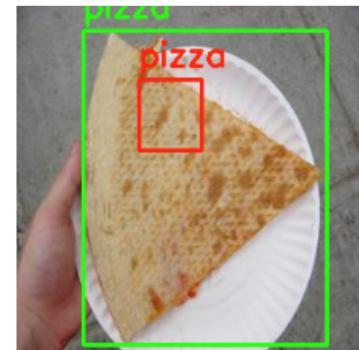
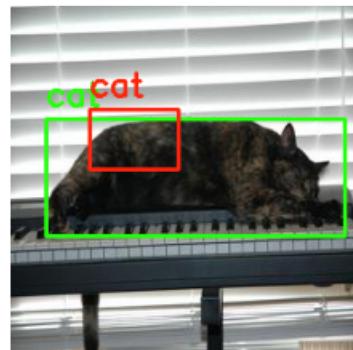
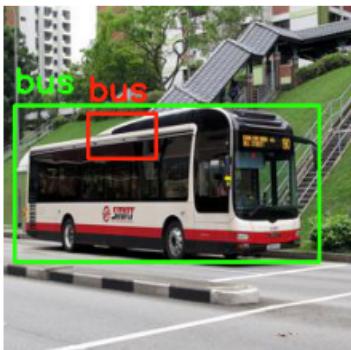
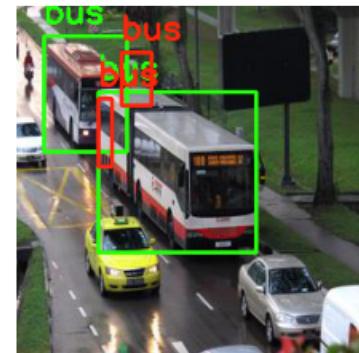
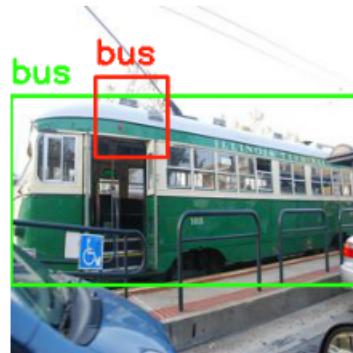
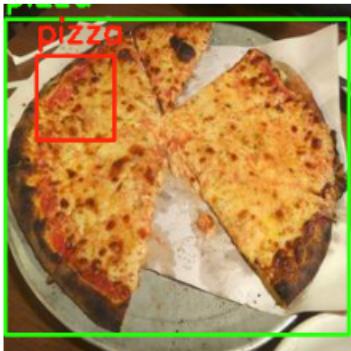
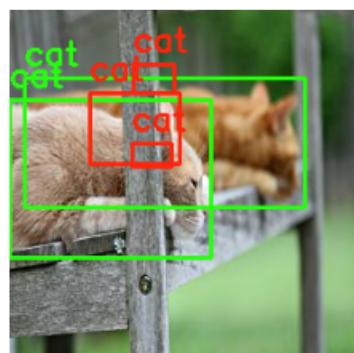
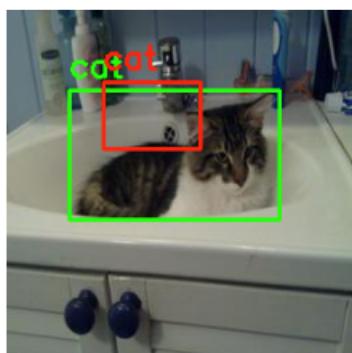
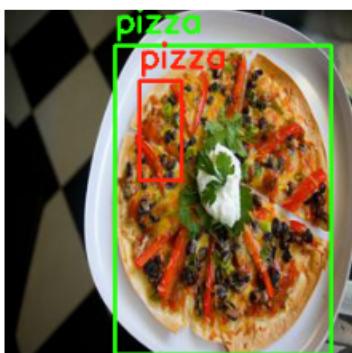
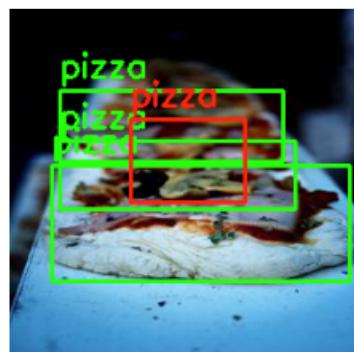
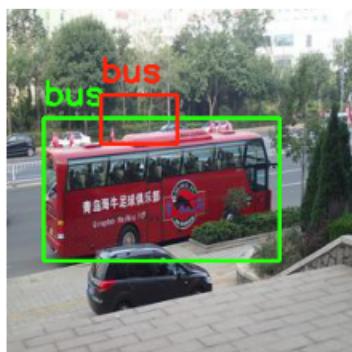
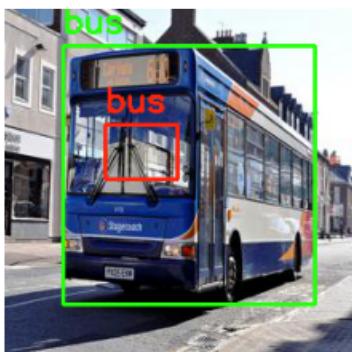
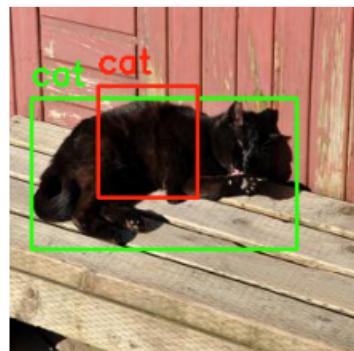
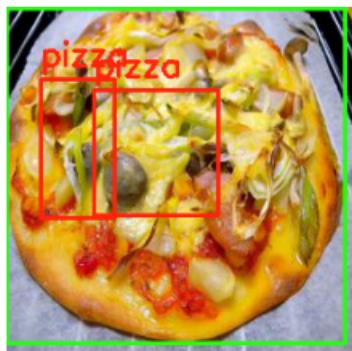
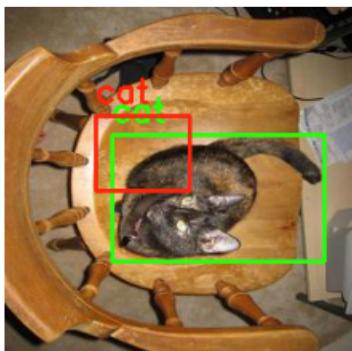


Figure 3: For testing

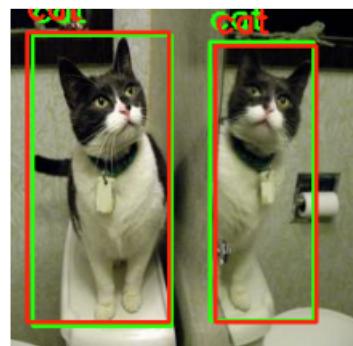
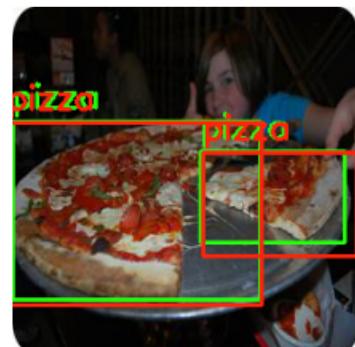
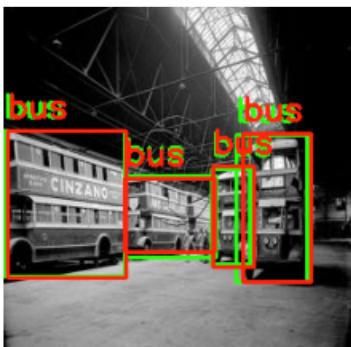
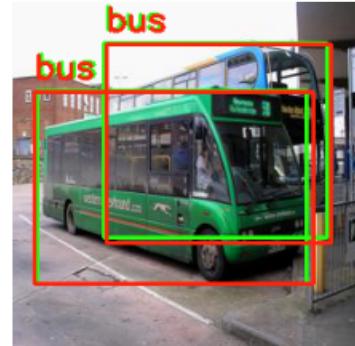
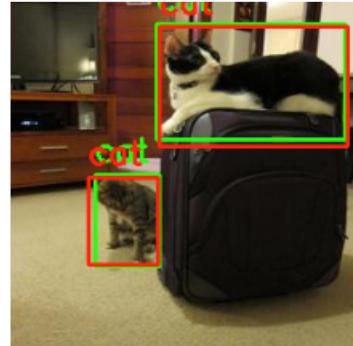
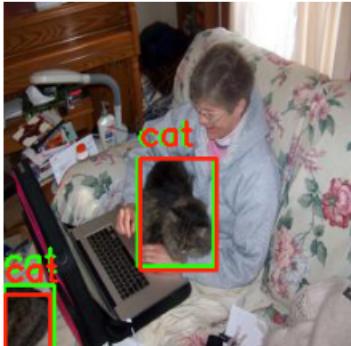
2.2.2 For Output of YOLO detector

Below are 18 images from the test dataset along with their ground truth bounding box in green and predicted bounding box in red.





Below are 9 images from the training dataset along with their ground truth bounding box in green and predicted bounding box in red.



3 Conclusion

Quantitatively demonstrating the performance of a multi-instance object detector is difficult. Therefore I have plotted the predicted bounding box and the ground truth bounding box for the images in the test and training dataset.

As you can observe in the plotted images for the test dataset, the model can classify the objects really well. The predicted bounding boxes for images having only one ground truth object are almost adequate. However, the model really struggles when there are multiple ground truth objects in the image. In these cases, the model often predicts an incorrect number of objects. Even when the model accurately predicts the correct number of objects, the predicted bounding boxes for them leave a lot to be desired.

I also plotted images containing multiple objects from the training dataset. The model performs really well in classifying as well as predicting the bounding boxes for these images.

Based on the model predictions on the images in the testing and training datasets, its clear to see that the model does learn well. However, there is a degree of overfitting as the results on the training images are far superior to the results on the test images.

In [1]:

```
#Referred to the COCO API Github repo
#importing the Libraries
from pycocotools.coco import COCO
import os
import time
import random
import requests
import matplotlib.pyplot as plt
import numpy as np
import skimage.io as io
from PIL import Image
import cv2
import json
```


In [2]:

```
#Function to download the dataset
def download_dataset(class_list,coco,dtype):
    catIds = coco.getCatIds(class_list) #Gets the ids for the pizza, bus and cat classes
    categories = coco.loadCats(catIds) #Gets the id,category,supercategory name for the
    categories.sort(key = lambda x:x['id']) #sorts the hashmap in the order of ids

    coco_labels_inverse = {} #Stores a hashmap where the key is the category id and the
    for idx, in_class in enumerate(class_list):
        for c in categories:
            if c['name'] == in_class:
                coco_labels_inverse[c['id']] = idx

    set_all = set()
    for catId in catIds:
        category = class_list[coco_labels_inverse[catId]]
        imgId = coco.getImgIds(catIds=catId ) #Getting all the ids of the images of the
        set_all = set_all.union(set(imgId))

    imgIds = list(set_all) #contains all the image ids of images containing an object in
    print("The total number of images is "+str(len(imgIds)))

    total_images = 0
    dictionary = {} #stores the annotations
    for i in range(len(imgIds)):
        img_url = coco.loadImgs(imgIds[i])[0] #url of the image
        img = io.imread(img_url['coco_url']) #The actual image
        im = Image.fromarray(img) #Converting it to PIL object
        im1 = im.resize((256,256)) #Resizing it

        annIds = coco.getAnnIds(img_url['id'],iscrowd=False) #Gets the annotation ids
        anns = coco.loadAnns(annIds) #gets the actual annotations
        array_info = []
        for k in range(len(anns)):
            if(anns[k]["area"]>4096 and anns[k]["category_id"] in catIds): #Checking if
                #Resizing the bounding box coordinates
                [x1,y1,w1,h1] = anns[k]["bbox"]
                x = x1*256.0/im.size[0]
                w = w1*256.0/im.size[0]
                y = y1*256.0/im.size[1]
                h = h1*256.0/im.size[1]
                cl = class_list[coco_labels_inverse[anns[k]["category_id"]]]
                array_info.append([[x,y,w,h],cl])
            else:
                continue

        if total_images==0 and len(array_info)>0: #Creating the directories
            dataset = os.path.join("Dataset",dtype)
            os.makedirs(dataset)

        if len(array_info)>0: #If the image contains atleast one object of area 4096 bel
            im1.save(os.path.join(dataset,str(total_images)+".jpeg")) #Saving the image
            dictionary[os.path.join(dataset,str(total_images)+".jpeg")] = array_info #So
            total_images = total_images+1

    print("The total number of images with a dominant object is "+str(total_images))
```

```
    return dictionary
In [3]:
```

```
coco1=COCO("instances_train2014.json")
coco2=COCO("instances_val2014.json")
class_list = ["pizza","bus","cat"]
```

```
loading annotations into memory...
Done (t=11.12s)
creating index...
index created!
loading annotations into memory...
Done (t=5.64s)
creating index...
index created!
```

```
In [4]:
```

```
#For the training dataset
train_dict = download_dataset(class_list,coco1,"train")
json1 = json.dumps(train_dict) #Creating a JSON object
f = open("dict_train.json","w") #opening file for writing
f.write(json1) #writing
f.close() #closing
```

```
The total number of images is 7799
The total number of images with a dominant object is 6883
```

```
In [5]:
```

```
#For the validation dataset
test_dict = download_dataset(class_list,coco2,"test")
json2 = json.dumps(test_dict) #Creating a JSON object
f = open("dict_test.json","w") #opening file for writing
f.write(json2) #writing
f.close() #closing
```

```
The total number of images is 3940
The total number of images with a dominant object is 3491
```

In [26]:

```
#Number of training images having "n" annotations
a = {}
for i in range(15):
    a[i] = 0
for loc in train_dict:
    a[len(train_dict[loc])] +=1
a
```

Out[26]:

```
{0: 0,
 1: 5578,
 2: 953,
 3: 237,
 4: 80,
 5: 18,
 6: 6,
 7: 5,
 8: 4,
 9: 0,
 10: 0,
 11: 0,
 12: 0,
 13: 1,
 14: 1}
```

In [28]:

```
#Number of testing images having "n" annotations
b = {}
for i in range(14):
    b[i] = 0
for loc in test_dict:
    b[len(test_dict[loc])] +=1
b
```

Out[28]:

```
{0: 0,
 1: 2889,
 2: 435,
 3: 104,
 4: 42,
 5: 7,
 6: 6,
 7: 2,
 8: 2,
 9: 3,
 10: 0,
 11: 0,
 12: 0,
 13: 1}
```

In []:

In [1]:

```
#importing the libraries
import torch
import torch.nn as nn
from torchvision import datasets,models,transforms,ops
from PIL import Image
from torch.utils.data import Dataset,DataLoader
import os
import time
from tqdm import tqdm
import json
import matplotlib.pyplot as plt
import numpy as np
import cv2
import pandas as pd
```


In [2]:

```

def convert_to_bbox_cord(yolo_tensor_aug):
    num_cells_along_width = 8
    yolo_interval = 32
    #Keeping track of index for which anchor box is best for all 64 cells
    best_ancbox_for_cell = {cx : None for cx in range(64)}
    for cx in range(yolo_tensor_aug.shape[0]):
        curr_best = 0
        cell_tensor = yolo_tensor_aug[cx] #Shape (5x9)
        #Finding the anchor box index for which the object has highest probability of being
        for ax in range(cell_tensor.shape[0]):
            if cell_tensor[ax][0] > cell_tensor[curr_best][0]:
                curr_best = ax
        best_ancbox_for_cell[cx] = curr_best

    #Sorting this dictionary in descending order of probability of a cell having an object
    sorted_icx = sorted(best_ancbox_for_cell,key=lambda x: yolo_tensor_aug[x,best_ancbox_for_cell[x]])
    retained_cells_idxs = sorted_icx[:5]

    #Extracting the labels and bounding boxes from the top 5 cells
    labels = []
    bbox = []
    for cx in retained_cells_idxs:
        yolo_vector = yolo_tensor_aug[cx,best_ancbox_for_cell[cx],:] #Extracting the yolo vector
        class_label_prob_all = nn.Softmax(dim=0)(yolo_vector[5:]) #Applying softmax to the class probabilities
        class_label_prob = class_label_prob_all[:-1]
        if yolo_vector[0]<0.25:
            continue
        else:
            #Extracting the label
            labels.append(torch.argmax(class_label_prob).item())

            #Extracting the height and width of the bounding box
            h = yolo_vector[3].item()*yolo_interval
            w = yolo_vector[4].item()*yolo_interval

            #Extracting the center coordinate of the bounding box wrt the cell
            del_x = yolo_vector[1].item()
            del_y = yolo_vector[2].item()

            #Getting the cell coordinates
            cell_row_idx = cx//num_cells_along_width
            cell_col_idx = cx%num_cells_along_width

            #Finding the yolo cell center 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

            #Finding the center coordinates of the bounding box
            bb_height_center = del_x*yolo_interval + cell_center_j
            bb_width_center = del_y*yolo_interval + cell_center_i

            #Finding top left and right of the bounding box
            x = bb_width_center - w/2.0
            y = bb_height_center - h/2.0
            bbox.append([x,y,x+w,y+h])

    return torch.Tensor(bbox),torch.Tensor(labels)

```


In [3]:

```
class IndexedDataset(Dataset):

    def __init__(self, dir_path):
        self.dir_path = dir_path

        if os.path.basename(self.dir_path) == 'train': #transforms for the train dataset
            self.transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])
            ])
            f = open('Dataset/dict_train.json')
            self.bbox_data = json.load(f)
            f.close()
            self.loc = "Dataset\\train\\"

        elif os.path.basename(self.dir_path) == 'test': #transforms for the test dataset
            self.transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])
            ])
            f = open('Dataset/dict_test.json')
            self.bbox_data = json.load(f)
            f.close()
            self.loc = "Dataset\\test\\"

        image_filenames = []
        for (dirpath, dirnames, filenames) in os.walk(dir_path): #Saving all the image l
            image_filenames += [os.path.join(dirpath, file) for file in filenames]
        self.image_filenames = image_filenames
        self.labels_map = {"bus" : 0, "cat": 1, "pizza" : 2} #Creating hashmap of the cl

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

    def convert_to_yolo_tensor(self,bbox,label):
        #bbox is of shape (number_of_objects x 4)
        #label is of shape (number_of_objects)

        yolo_interval = 32
        cell_height = yolo_interval
        cell_width = yolo_interval
        num_cells_along_width = 8
        num_cells_along_height = 8
        yolo_tensor = torch.zeros(64,5,8)
        yolo_tensor_aug = torch.zeros(64,5,9)

        for ix in range(len(bbox)):
            #Center coordinates of the bounding box
            bb_height_center = (bbox[ix,1] + bbox[ix,3])/2.0
            bb_width_center = (bbox[ix,0] + bbox[ix,2])/2.0

            #height and width bounding box
            obj_bb_height = bbox[ix,3] - bbox[ix,1]
            obj_bb_width = bbox[ix,2] - bbox[ix,0]

            #Finding the index of the yolo cell
            cell_row_idx = int(bb_height_center / yolo_interval)
            cell_col_idx = int(bb_width_center / yolo_interval)
```

```

#Center coordinates of the yolo cell
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

#Center coordinates of the bounding box wrt to the yolo cell
del_x = (bb_height_center - cell_center_j) / yolo_interval
del_y = (bb_width_center - cell_center_i) / yolo_interval

#Height and width of the bounding box wrt to the yolo cell
bh = obj_bb_height/ yolo_interval
bw = obj_bb_width/ yolo_interval

#Finding the correct anchor box
ratio = obj_bb_height/obj_bb_width
if ratio <= 0.2:
    anc_box_i = 0
elif 0.2 < ratio <= 0.5:
    anc_box_i = 1
elif 0.5 < ratio <= 1.5:
    anc_box_i = 2
elif 1.5 < ratio <= 4.0:
    anc_box_i = 3
elif ratio > 4.0:
    anc_box_i = 4

#Constructing the yolo vector
yolo_vector = torch.FloatTensor([1,del_x.item(), del_y.item(), bh.item(), bw.item()])
yolo_vector[5 + int(label[idx].item())] = 1

#Inserting the yolo vector into the yolo tensor
cell_index = cell_row_idx * num_cells_along_width + cell_col_idx
yolo_tensor[cell_index, anc_box_i] = yolo_vector

## If no object is present, throw all the probability mass into the extra 9th element
yolo_tensor_aug[:, :, :-1] = yolo_tensor[:, :, :]
for cx in range(64): # Over all the yolo cells
    for ax in range(5): # Over all the anchor boxes
        if yolo_tensor_aug[cx,ax,0] == 0:
            yolo_tensor_aug[cx,ax,-1] = 1

return yolo_tensor_aug

def __getitem__(self, idx):
    img_name = self.image_filenames[idx]
    image = Image.open(img_name).convert('RGB')
    image = self.transform(image)
    bbox_loc = self.loc + os.path.basename(img_name)
    bbox = self.bbox_data[bbox_loc]
    cl = []
    bb = []
    for i in range(len(bbox)):
        cl.append(self.labels_map[bbox[i][1]])
        b = bbox[i][0]
        bb.append(b)
    #x and y have traditional math notations
    #Changing the last two variables from w,h to x2,y2
    bb[:,2] = bb[:,2]+bb[:,0]
    bb[:,3] = bb[:,3]+bb[:,1]

    yolo_tensor = self.convert_to_yolo_tensor(bb,torch.Tensor(cl))

```

In [4]: `return image, yolo_tensor`

```
#Function to Plot the image with the ground truth annotation
def plot_image_ground(dataset,index):
    img = dataset[index][0] #Getting the image
    i = np.transpose(np.asarray(img*127.5 + 127.5).astype(int),(1,2,0)) #converting the image
    ci = np.ascontiguousarray(i, dtype=np.uint8) #Making the array contiguous
    labels_map = {0:"bus", 1: "cat", 2: "pizza"}

    #Getting the ground truth bounding box and Labels
    yolo_tensor = dataset[index][1]
    bbox, labels = convert_to_bbox_cord(yolo_tensor)

    #Plotting the ground truth bounding boxes
    for i in range(len(labels)):
        cl = labels_map[int(labels[i].item())] #Class of the image
        [x1,y1,x2,y2] = bbox[i] #Getting the ground truth bounding box coordinates
        ci = cv2.rectangle(ci,(int(x1),int(y1)),(int(x2),int(y2)),(36,255,12),2) #drawing the bounding box
        ci = cv2.putText(ci,cl,(int(x1),int(y1-10)),cv2.FONT_HERSHEY_SIMPLEX,0.8,(36,255,12))

    return ci.astype(int) #returns the image with the bounding box
```

In [5]:

```
train_dataset = IndexedDataset("Dataset/train")
print(len(train_dataset))
```

6883

In [6]:

```
#Plotting 9 images along with ground truth annotations
fig = plt.figure(figsize = (15,10))

ax11 = fig.add_subplot(3,3,1)
ax12 = fig.add_subplot(3,3,2)
ax13 = fig.add_subplot(3,3,3)

ax11.axis('off')
ax12.axis('off')
ax13.axis('off')

ax11.imshow(plot_image_ground(train_dataset,1))
ax12.imshow(plot_image_ground(train_dataset,2))
ax13.imshow(plot_image_ground(train_dataset,6))

ax21 = fig.add_subplot(3,3,4)
ax22 = fig.add_subplot(3,3,5)
ax23 = fig.add_subplot(3,3,6)

ax21.axis('off')
ax22.axis('off')
ax23.axis('off')

ax21.imshow(plot_image_ground(train_dataset,801))
ax22.imshow(plot_image_ground(train_dataset,800))
ax23.imshow(plot_image_ground(train_dataset,806))

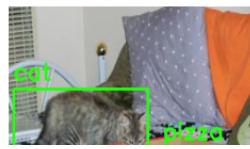
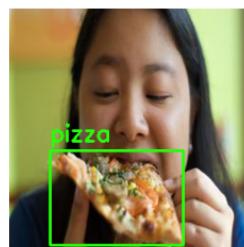
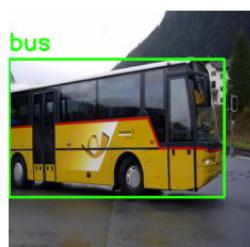
ax31 = fig.add_subplot(3,3,7)
ax32 = fig.add_subplot(3,3,8)
ax33 = fig.add_subplot(3,3,9)

ax31.axis('off')
ax32.axis('off')
ax33.axis('off')

ax31.imshow(plot_image_ground(train_dataset,1899))
ax32.imshow(plot_image_ground(train_dataset,1902))
ax33.imshow(plot_image_ground(train_dataset,804))
```

Out[6]:

```
<matplotlib.image.AxesImage at 0x25fc4cfe670>
```



In [1]:

```
import torch
import torch.nn as nn
from torchvision import datasets,models,transforms,ops
from PIL import Image
from torch.utils.data import Dataset,DataLoader
import os
import time
from tqdm import tqdm
import json
import matplotlib.pyplot as plt
import numpy as np
import cv2
import pandas as pd
import seaborn as sn
```


In [2]:

```

def convert_to_bbox_cord(yolo_tensor_aug):
    num_cells_along_width = 8
    yolo_interval = 32
    #Keeping track of index for which anchor box is best for all 64 cells
    best_ancbox_for_cell = {cx : None for cx in range(64)}
    for cx in range(yolo_tensor_aug.shape[0]):
        curr_best = 0
        cell_tensor = yolo_tensor_aug[cx] #Shape (5x9)
        #Finding the anchor box index for which the object has highest probability of being
        for ax in range(cell_tensor.shape[0]):
            if cell_tensor[ax][0] > cell_tensor[curr_best][0]:
                curr_best = ax
        best_ancbox_for_cell[cx] = curr_best

    #Sorting this dictionary in descending order of probability of a cell having an object
    sorted_icx = sorted(best_ancbox_for_cell, key=lambda x: yolo_tensor_aug[x,best_ancbox_for_cell[x]])
    retained_cells_idxs = sorted_icx[:5]

    #Extracting the labels and bounding boxes from the top 5 cells
    labels = []
    bbox = []
    for cx in retained_cells_idxs:
        yolo_vector = yolo_tensor_aug[cx,best_ancbox_for_cell[cx],:] #Extracting the yolo vector
        class_label_prob_all = nn.Softmax(dim=0)(yolo_vector[5:]) #Applying softmax to the class probabilities
        class_label_prob = class_label_prob_all[:-1]
        if yolo_vector[0]<0.25:
            continue
        else:
            #Extracting the label
            labels.append(torch.argmax(class_label_prob).item())

            #Extracting the height and width of the bounding box
            h = yolo_vector[3].item()*yolo_interval
            w = yolo_vector[4].item()*yolo_interval

            #Extracting the center coordinate of the bounding box wrt the cell
            del_x = yolo_vector[1].item()
            del_y = yolo_vector[2].item()

            #Getting the cell coordinates
            cell_row_idx = cx//num_cells_along_width
            cell_col_idx = cx%num_cells_along_width

            #Finding the yolo cell center 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

            #Finding the center coordinates of the bounding box
            bb_height_center = del_x*yolo_interval + cell_center_j
            bb_width_center = del_y*yolo_interval + cell_center_i

            #Finding top left and right of the bounding box
            x = bb_width_center - w/2.0
            y = bb_height_center - h/2.0
            bbox.append([x,y,x+w,y+h])

    return torch.Tensor(bbox),torch.Tensor(labels)

```


In [3]:

```

class IndexedDataset(Dataset):

    def __init__(self, dir_path):
        self.dir_path = dir_path

        if os.path.basename(self.dir_path) == 'train': #transforms for the train dataset
            self.transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])
            ])
            f = open('Dataset/dict_train.json')
            self.bbox_data = json.load(f)
            f.close()
            self.loc = "Dataset\\train\\"

        elif os.path.basename(self.dir_path) == 'test': #transforms for the test dataset
            self.transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize([0.5,0.5,0.5],[0.5,0.5,0.5])
            ])
            f = open('Dataset/dict_test.json')
            self.bbox_data = json.load(f)
            f.close()
            self.loc = "Dataset\\test\\"

        image_filenames = []
        for (dirpath, dirnames, filenames) in os.walk(dir_path): #Saving all the image l
            image_filenames += [os.path.join(dirpath, file) for file in filenames]
        self.image_filenames = image_filenames
        self.labels_map = {"bus" : 0, "cat": 1, "pizza" : 2} #Creating hashmap of the cl

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

    def convert_to_yolo_tensor(self,bbox,label):
        #bbox is of shape (number_of_objects x 4)
        #label is of shape (number_of_objects)

        yolo_interval = 32
        cell_height = yolo_interval
        cell_width = yolo_interval
        num_cells_along_width = 8
        num_cells_along_height = 8
        yolo_tensor = torch.zeros(64,5,8)
        yolo_tensor_aug = torch.zeros(64,5,9)

        for ix in range(len(bbox)):
            #Center coordinates of the bounding box
            bb_height_center = (bbox[ix,1] + bbox[ix,3])/2.0
            bb_width_center = (bbox[ix,0] + bbox[ix,2])/2.0

            #height and width bounding box
            obj_bb_height = bbox[ix,3] - bbox[ix,1]
            obj_bb_width = bbox[ix,2] - bbox[ix,0]

            #Finding the index of the yolo cell
            cell_row_idx = int(bb_height_center / yolo_interval)
            cell_col_idx = int(bb_width_center / yolo_interval)

```

```

#Center coordinates of the yolo cell
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

#Center coordinates of the bounding box wrt to the yolo cell
del_x = (bb_height_center - cell_center_j) / yolo_interval
del_y = (bb_width_center - cell_center_i) / yolo_interval

#Height and width of the bounding box wrt to the yolo cell
bh = obj_bb_height/ yolo_interval
bw = obj_bb_width/ yolo_interval

#Finding the correct anchor box
ratio = obj_bb_height/obj_bb_width
if ratio <= 0.2:
    anc_box_i = 0
elif 0.2 < ratio <= 0.5:
    anc_box_i = 1
elif 0.5 < ratio <= 1.5:
    anc_box_i = 2
elif 1.5 < ratio <= 4.0:
    anc_box_i = 3
elif ratio > 4.0:
    anc_box_i = 4

#Constructing the yolo vector
yolo_vector = torch.FloatTensor([1,del_x.item(), del_y.item(), bh.item(), bw.item()])
yolo_vector[5 + int(label[idx].item())] = 1

#Inserting the yolo vector into the yolo tensor
cell_index = cell_row_idx * num_cells_along_width + cell_col_idx
yolo_tensor[cell_index, anc_box_i] = yolo_vector

## If no object is present, throw all the probability mass into the extra 9th element
yolo_tensor_aug[:, :, :-1] = yolo_tensor[:, :, :]
for cx in range(64): # Over all the yolo cells
    for ax in range(5): # Over all the anchor boxes
        if yolo_tensor_aug[cx,ax,0] == 0:
            yolo_tensor_aug[cx,ax,-1] = 1

return yolo_tensor_aug

def __getitem__(self, idx):
    img_name = self.image_filenames[idx]
    image = Image.open(img_name).convert('RGB')
    image = self.transform(image)
    bbox_loc = self.loc + os.path.basename(img_name)
    bbox = self.bbox_data[bbox_loc]
    cl = []
    bb = []
    for i in range(len(bbox)):
        cl.append(self.labels_map[bbox[i][1]])
        b = bbox[i][0]
        bb.append(b)
    #x and y have traditional math notations
    #Changing the last two variables from w,h to x2,y2
    bb[:,2] = bb[:,2]+bb[:,0]
    bb[:,3] = bb[:,3]+bb[:,1]

    yolo_tensor = self.convert_to_yolo_tensor(bb,torch.Tensor(cl))

```

```
In [4]: return image, yolo_tensor
```

```
#Function to Plot the image with the ground truth annotation
def plot_image_ground(model,dataset,index,device):
    img = dataset[index][0] #Getting the image
    i = np.transpose(np.asarray(img*127.5 + 127.5).astype(int),(1,2,0)) #converting the image
    ci = np.ascontiguousarray(i, dtype=np.uint8) #Making the array contiguous
    labels_map = {0:"bus", 1: "cat", 2: "pizza"}

    #Getting the ground truth bounding box and Labels
    yolo_tensor = dataset[index][1]
    bbox, labels = convert_to_bbox_cord(yolo_tensor)

    #Plotting the ground truth bounding boxes
    for i in range(len(labels)):
        cl = labels_map[int(labels[i].item())] #Class of the image
        [x1,y1,x2,y2] = bbox[i] #Getting the ground truth bounding box coordinates
        ci = cv2.rectangle(ci,(int(x1),int(y1)),(int(x2),int(y2)),(36,255,12),2) #drawing the box
        ci = cv2.putText(ci,cl,(int(x1),int(y1-10)),cv2.FONT_HERSHEY_SIMPLEX,0.8,(36,255,12))

    #Getting the predicted bounding boxes
    pred_yolo_tensor = model(img.unsqueeze(dim=0).to(device)).squeeze()
    pbbox, plabels = convert_to_bbox_cord(pred_yolo_tensor.cpu())

    #Plotting the predicted bounding boxes
    for i in range(len(plabels)):
        pcl = labels_map[int(plabels[i].item())] #Class of the image
        [x1,y1,x2,y2] = pbbox[i] #Getting the predicted bounding box coordinates
        ci = cv2.rectangle(ci,(int(x1),int(y1)),(int(x2),int(y2)),(255,36,12),2) #drawing the box
        ci = cv2.putText(ci,cl,(int(x1),int(y1-10)),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,36,12))

    plt.imshow(ci) #plotting the image with the bounding box
    plt.axis("off")
```

In [5]:

```
class ResnetBlock(nn.Module):
    def __init__(self, input_nc, output_nc, stride):
        super().__init__()
        self.conv1 = nn.Conv2d(input_nc, output_nc, kernel_size=3, stride=stride, padding=1)
        self.bn1 = nn.BatchNorm2d(output_nc)
        self.relu1 = nn.ReLU(True)
        self.conv2 = nn.Conv2d(output_nc, output_nc, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(output_nc)
        self.conv3 = nn.Conv2d(input_nc, output_nc, kernel_size=3, stride=stride, padding=1)
        self.bn3 = nn.BatchNorm2d(output_nc)
        self.relu3 = nn.ReLU(True)
        self.down = True if (stride!=1 or input_nc!=output_nc) else False

    def forward(self, x):
        identity = x
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu1(x)

        x = self.conv2(x)
        x = self.bn2(x)

        if self.down:
            identity = self.conv3(identity)
            identity = self.bn3(identity)

        x = x+identity
        x = self.relu3(x)

    return x
```

In [6]:

```
class HW6Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1)
        self.block1 = ResnetBlock(64, 64, 1)
        self.block2 = ResnetBlock(64, 64, 1)
        self.block3 = ResnetBlock(64, 128, 2)
        self.block4 = ResnetBlock(128, 128, 1)
        self.block5 = ResnetBlock(128, 256, 2)
        self.block6 = ResnetBlock(256, 256, 1)
        self.block7 = ResnetBlock(256, 512, 2)
        self.block8 = ResnetBlock(512, 512, 1)
        self.fc = nn.Linear(512*8*8, 8*8*5*9)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)
        xc = self.block1(x)
        xc = self.block2(xc)
        xc = self.block3(xc)
        xc = self.block4(xc)
        xc = self.block5(xc)
        xc = self.block6(xc)
        xc = self.block7(xc)
        xc = self.block8(xc)
        xc = xc.view(xc.shape[0], -1)
        xc = self.fc(xc)
        xc = xc.view(xc.shape[0], 64, 5, 9)
        return xc
```


In [7]:

```

def calculate_loss(data_loader,criterion1,criterion2,criterion3,net,train,optimizer):
    running_loss = 0.0
    running_loss1 = 0.0
    running_loss2 = 0.0
    running_loss3 = 0.0
    for i, data in enumerate(data_loader):
        inputs, yolo_tensor_aug = data
        inputs = inputs.to(device)
        yolo_tensor_aug = yolo_tensor_aug.to(device)

        #Passing the image through the model to get the predicted yolo tensor
        if train:
            optimizer.zero_grad()
        predictions_aug = net(inputs)

        #Calculating the losses
        loss = torch.tensor(0.0, requires_grad=True).float().to(device) #Overall Loss

        #Only going ahead with the yolo vectors where there is an object predicted or actual
        idx1 = torch.where(predictions_aug.view(-1,9)[:,0]>0)[0]
        idx2 = torch.where(yolo_tensor_aug.view(-1,9)[:,0]>0.5)[0]
        idx = list(set(torch.Tensor.tolist(idx1)).union(set(torch.Tensor.tolist(idx2)))) 

        if len(idx) == 0:
            continue

        #All the ground truth and predicted yolo vectors that satisfy the above condition
        pred_yolo_vector = predictions_aug.view(-1,9)[idx]
        target_yolo_vector = yolo_tensor_aug.view(-1,9)[idx]

        #Loss for the presence of the object (BCE)
        pred_obj = nn.Sigmoid()(pred_yolo_vector[:,0])
        target_obj = target_yolo_vector[:,0]
        loss1 = criterion1(pred_obj,target_obj)

        #Loss for the bounding box coordinates (MSE)
        pred_cor = pred_yolo_vector[:,1:5]
        target_cor = target_yolo_vector[:,1:5]
        loss2 = criterion2(pred_cor,target_cor)

        #Loss for the class of the object (CE)
        pred_cl = pred_yolo_vector[:,5:]
        target_cl = torch.max(target_yolo_vector[:,5:].data, 1)[1]
        loss3 = criterion3(pred_cl,target_cl)

        loss = loss1 + loss2 + loss3
        if train:
            loss.backward()
            optimizer.step()

        running_loss += loss.cpu().item()
        running_loss1 += loss1.cpu().item()
        running_loss2 += loss2.cpu().item()
        running_loss3 += loss3.cpu().item()
    
```

```
    return net,running_loss, running_loss1, running_loss2, running_loss3
In [8]:
```

```
def training(epochs,optimizer,criterion1,criterion2,criterion3,net,train_data_loader,test_data_loader,scheduler):
    train_losses = []
    train_losses1 = []
    train_losses2 = []
    train_losses3 = []

    test_losses = []
    test_losses1 = []
    test_losses2 = []
    test_losses3 = []

    for epoch in range(epochs):
        print("epoch: "+str(epoch))
        net.train()
        net,running_loss, running_loss1, running_loss2, running_loss3 = calculate_loss(train_data_loader,net,criterion1,criterion2,criterion3)
        train_losses.append(running_loss/len(train_data_loader))
        train_losses1.append(running_loss1/len(train_data_loader))
        train_losses2.append(running_loss2/len(train_data_loader))
        train_losses3.append(running_loss3/len(train_data_loader))
        print("[Training] total loss: %.3f BCE loss: %.3f MSE loss: %.3f CE loss: %.3f" % (total_loss, bce_loss, mse_loss, ce_loss))

        net.eval()
        net,test_loss, test_loss1, test_loss2, test_loss3 = calculate_loss(test_data_loader,net,criterion1,criterion2,criterion3)
        test_losses.append(running_loss/len(test_data_loader))
        test_losses1.append(running_loss1/len(test_data_loader))
        test_losses2.append(running_loss2/len(test_data_loader))
        test_losses3.append(running_loss3/len(test_data_loader))
        print("[Testing] total loss: %.3f BCE loss: %.3f MSE loss: %.3f CE loss: %.3f" % (total_loss, bce_loss, mse_loss, ce_loss))

        scheduler.step()

    return net, [train_losses,train_losses1,train_losses2,train_losses3], [test_losses,test_losses1,test_losses2,test_losses3]
```

```
In [9]:
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

```
Out[9]:
```

```
device(type='cuda')
```

In [10]:

```
train_dataset = IndexDataset("/content/drive/MyDrive/Dataset_HW6/train")
test_dataset = IndexDataset("/content/drive/MyDrive/Dataset_HW6/test")
train_dataloader = DataLoader(train_dataset, batch_size=96, shuffle=True, num_workers=96)
test_dataloader = DataLoader(test_dataset, batch_size=96, shuffle=True, num_workers=96)
print(len(train_dataset))
print(len(test_dataset))
```

6883

3491

```
/usr/local/lib/python3.9/dist-packages/torch/utils/data/dataloader.py:55
4: UserWarning: This DataLoader will create 96 worker processes in total.
Our suggested max number of worker in current system is 12, which is smaller than what this DataLoader is going to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the worker number to avoid potential slowness/freeze if necessary.
    warnings.warn(_create_warning_msg(
```

In [11]:

```
model = HW6Net()
model = model.to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.00001, betas = (0.9,0.99))
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=80, gamma=0.1)

criterion1 = nn.BCELoss()
criterion2 = nn.MSELoss()
criterion3 = nn.CrossEntropyLoss()
epochs = 100
```

In [12]:

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

Out[12]:

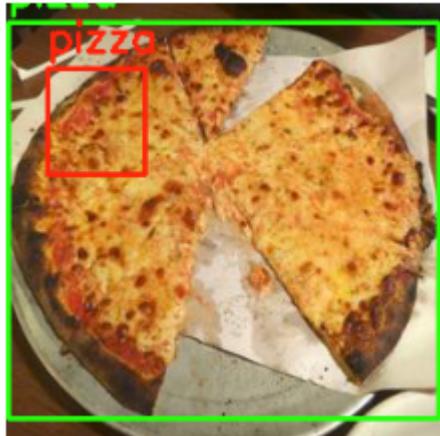
110105664

In [13]:

```
trained_model,train_losses,test_losses = training(epochs,optimizer,criterion1,criterion2
epoch: 0
[Training] total loss: 6.131 BCE loss: 1.063 MSE loss: 3.761 CE loss:
1.307
[Testing] total loss: 11.930 BCE loss: 2.068 MSE loss: 7.319 CE loss:
2.543
epoch: 1
[Training] total loss: 5.211 BCE loss: 0.954 MSE loss: 3.246 CE loss:
1.011
[Testing] total loss: 10.140 BCE loss: 1.857 MSE loss: 6.316 CE loss:
1.967
epoch: 2
[Training] total loss: 4.545 BCE loss: 0.880 MSE loss: 2.836 CE loss:
0.829
[Testing] total loss: 8.844 BCE loss: 1.712 MSE loss: 5.519 CE loss:
1.612
epoch: 3
[Training] total loss: 3.993 BCE loss: 0.818 MSE loss: 2.476 CE loss:
0.699
[Testing] total loss: 7.771 BCE loss: 1.593 MSE loss: 4.819 CE loss:
1.555
```

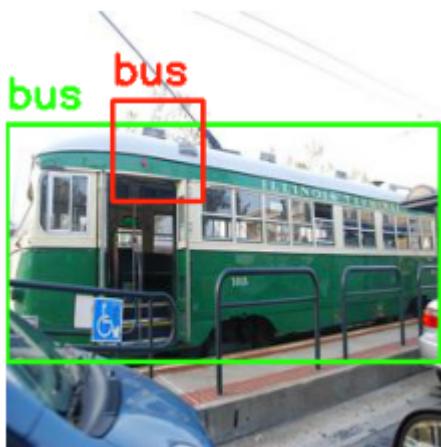
In [18]:

```
plot_image_ground(trained_model,test_dataset,2,device)
```



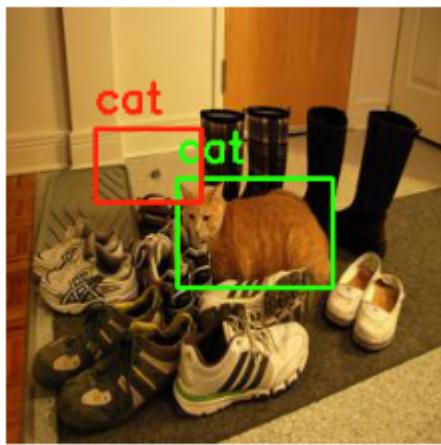
In [23]:

```
plot_image_ground(trained_model,test_dataset, 3,device)
```



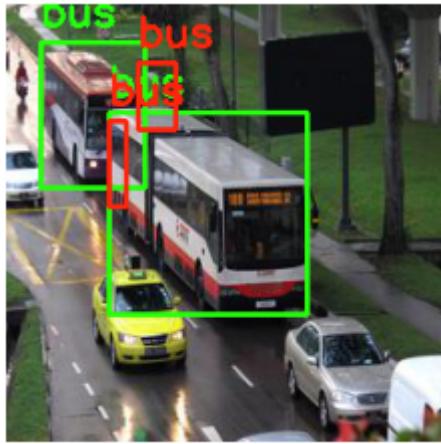
In [45]:

```
plot_image_ground(trained_model,test_dataset, 25,device)
```



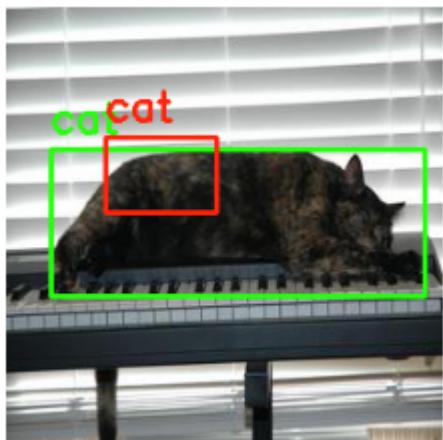
In [51]:

```
plot_image_ground(trained_model,test_dataset, 30,device)
```



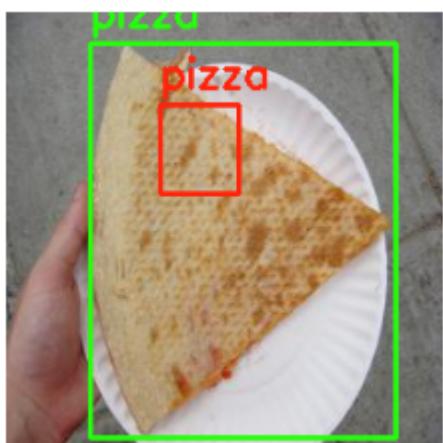
In [55]:

```
plot_image_ground(trained_model,test_dataset, 34,device)
```



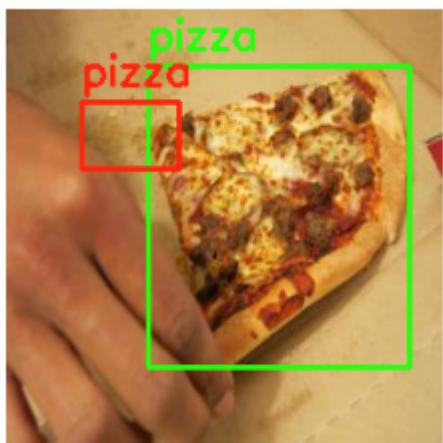
In [57]:

```
plot_image_ground(trained_model,test_dataset, 37,device)
```



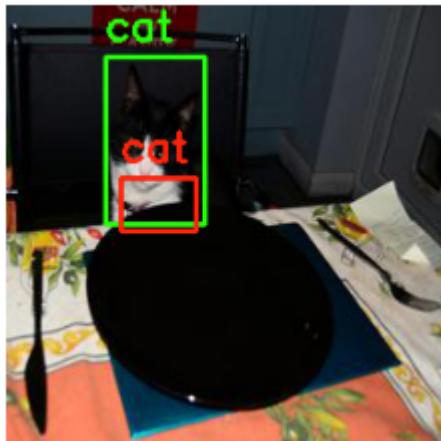
In [66]:

```
plot_image_ground(trained_model,test_dataset, 47,device)
```



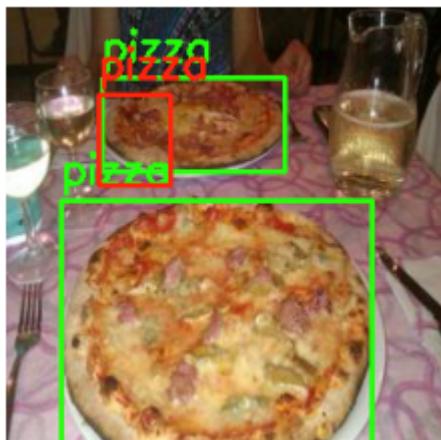
In [71]:

```
plot_image_ground(trained_model,test_dataset, 52,device)
```



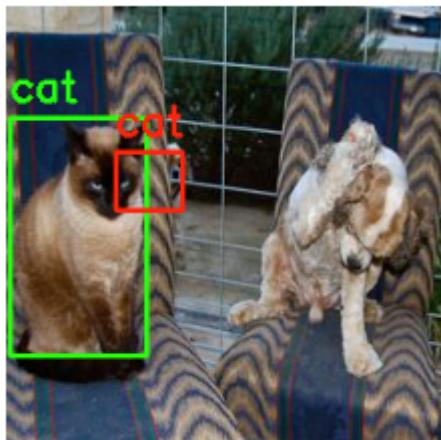
In [78]:

```
plot_image_ground(trained_model,test_dataset, 59,device)
```



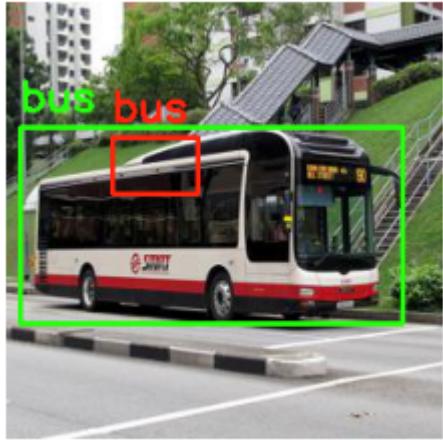
In [89]:

```
plot_image_ground(trained_model,test_dataset, 70,device)
```



In [90]:

```
plot_image_ground(trained_model,test_dataset, 71,device)
```



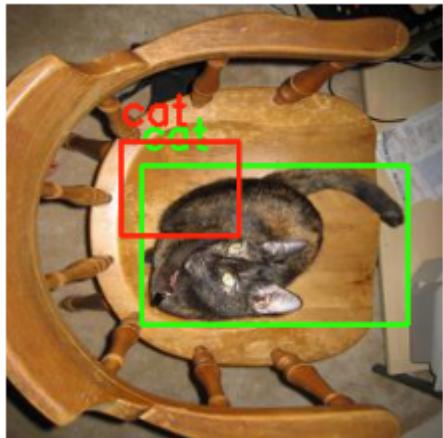
In [92]:

```
plot_image_ground(trained_model,test_dataset, 73,device)
```



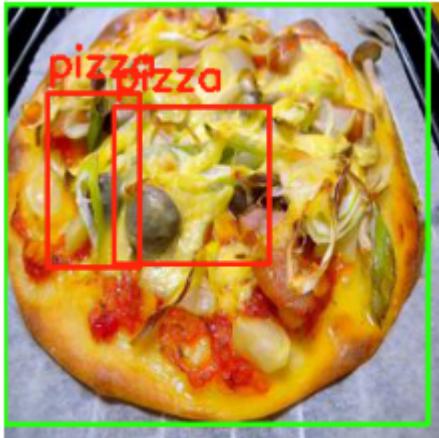
In [100]:

```
plot_image_ground(trained_model,test_dataset, 81,device)
```



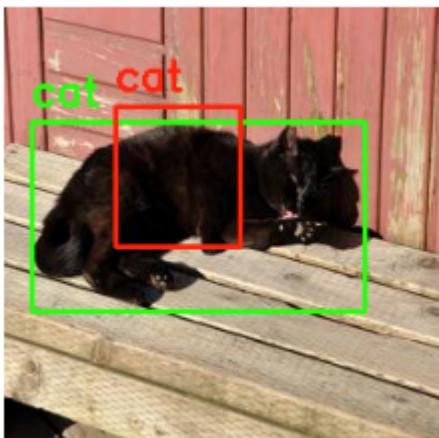
In [113]:

```
plot_image_ground(trained_model,test_dataset, 92,device)
```



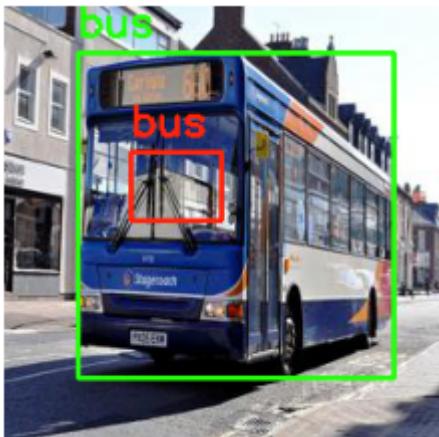
In [132]:

```
plot_image_ground(trained_model,test_dataset, 111,device)
```



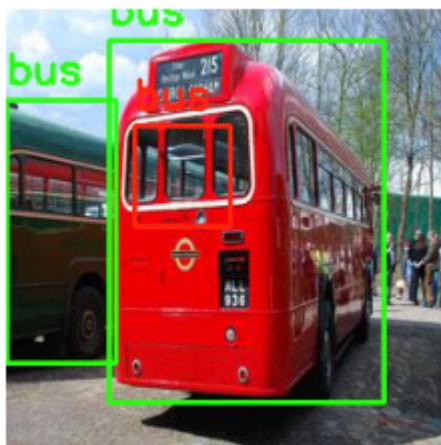
In [162]:

```
plot_image_ground(trained_model,test_dataset, 140,device)
```



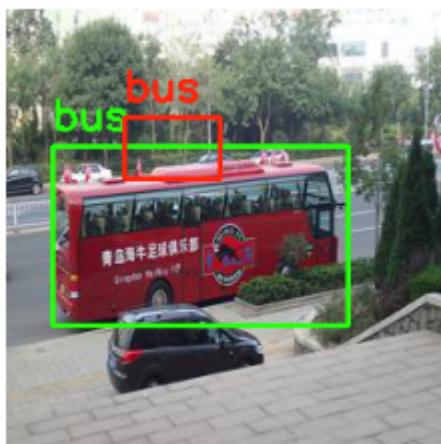
In [171]:

```
plot_image_ground(trained_model,test_dataset, 148,device)
```



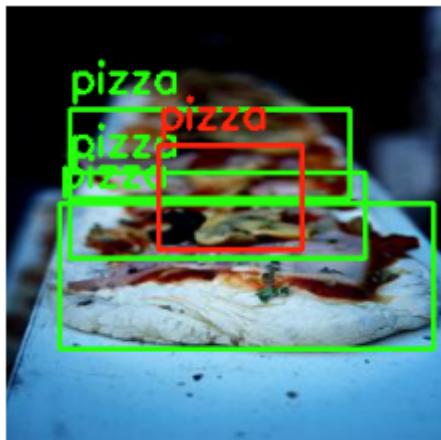
In [185]:

```
plot_image_ground(trained_model,test_dataset, 162,device)
```



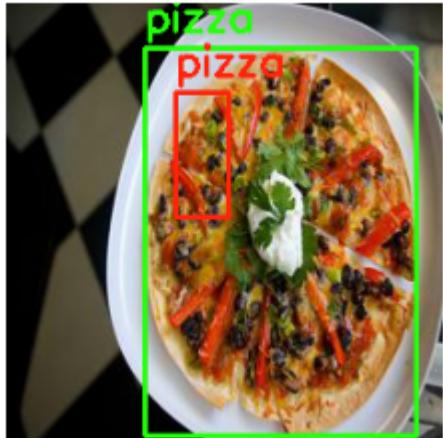
In [202]:

```
plot_image_ground(trained_model,test_dataset, 178,device)
```



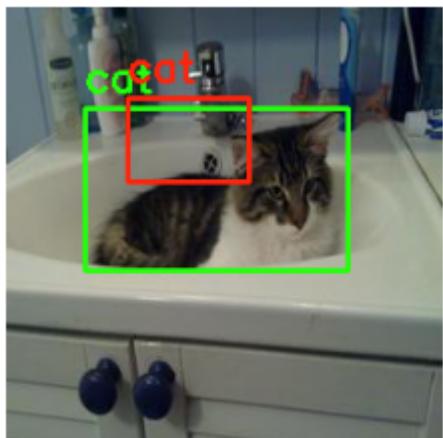
In [205]:

```
plot_image_ground(trained_model,test_dataset, 181,device)
```



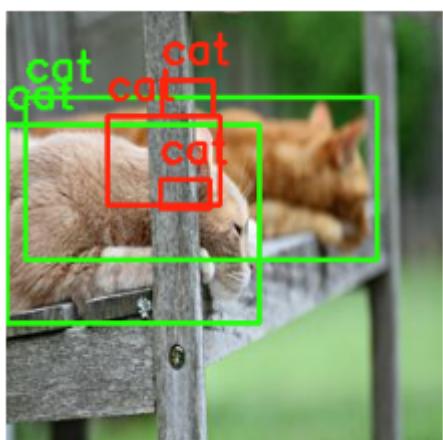
In [216]:

```
plot_image_ground(trained_model,test_dataset, 192,device)
```



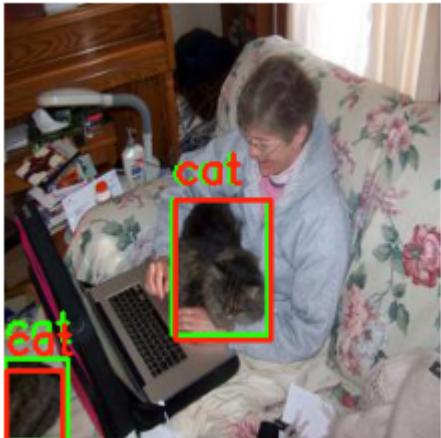
In [217]:

```
plot_image_ground(trained_model,test_dataset, 193,device)
```



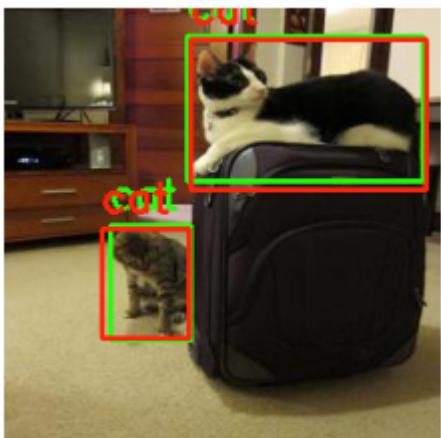
In [236]:

```
#TRAIN DATASET  
plot_image_ground(trained_model,train_dataset, 11,device)
```



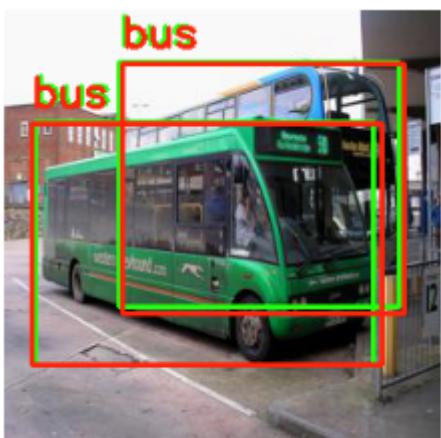
In [239]:

```
#TRAIN DATASET  
plot_image_ground(trained_model,train_dataset, 14,device)
```



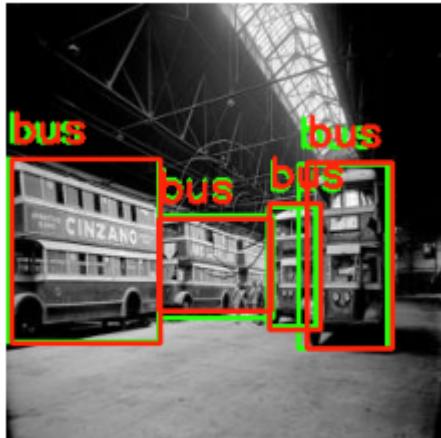
In [244]:

```
#TRAIN DATASET  
plot_image_ground(trained_model,train_dataset, 19,device)
```



In [249]:

```
#TRAIN DATASET  
plot_image_ground(trained_model,train_dataset, 24,device)
```



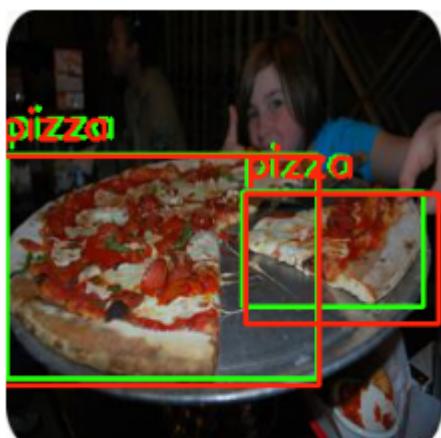
In [250]:

```
#TRAIN DATASET  
plot_image_ground(trained_model,train_dataset, 25,device)
```



In [251]:

```
#TRAIN DATASET  
plot_image_ground(trained_model,train_dataset, 26,device)
```



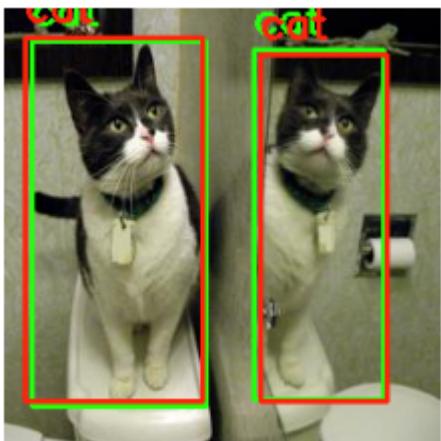
In [255]:

```
#TRAIN DATASET  
plot_image_ground(trained_model,train_dataset, 30,device)
```



In [263]:

```
#TRAIN DATASET  
plot_image_ground(trained_model,train_dataset, 38,device)
```



In [268]:

```
#TRAIN DATASET  
plot_image_ground(trained_model,train_dataset, 43,device)
```



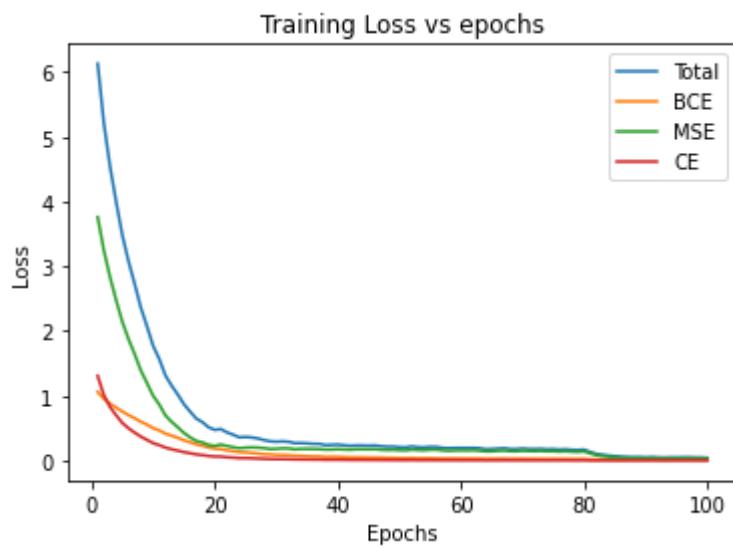
In [271]:

```
#Plotting the training Loss vs epochs for all the three models
epochs = np.arange(1,101)

plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training Loss vs epochs")
plt.plot(epochs,train_losses[0],label="Total")
plt.plot(epochs,train_losses[1],label="BCE")
plt.plot(epochs,train_losses[2],label="MSE")
plt.plot(epochs,train_losses[3],label="CE")
plt.legend(loc = "upper right")
```

Out[271]:

```
<matplotlib.legend.Legend at 0x7ff652e1aa60>
```



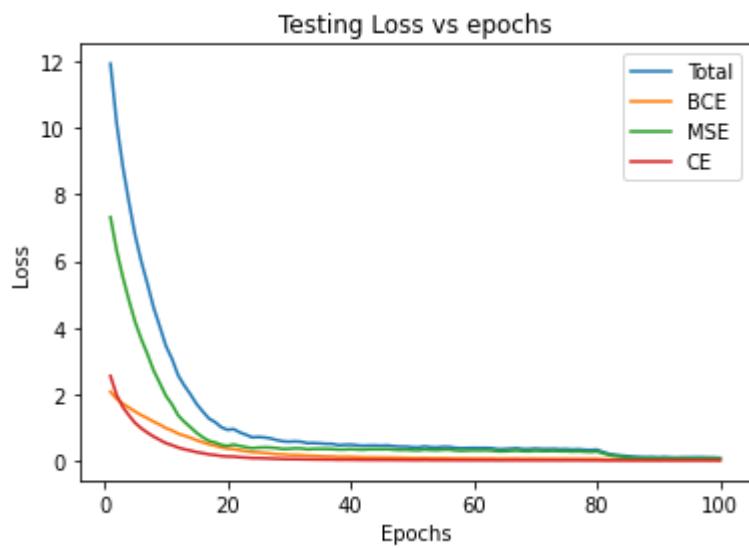
In [272]:

```
#Plotting the testing Loss vs epochs for all the three models
epochs = np.arange(1,101)

plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Testing Loss vs epochs")
plt.plot(epochs,test_losses[0],label="Total")
plt.plot(epochs,test_losses[1],label="BCE")
plt.plot(epochs,test_losses[2],label="MSE")
plt.plot(epochs,test_losses[3],label="CE")
plt.legend(loc = "upper right")
```

Out[272]:

<matplotlib.legend.Legend at 0x7ff652db4df0>



In []: