PURDUE UNIVERSITY

Elmore Family School of Electrical and Computer Engineering

Deep Learning

Homework 5

Adithya Sineesh

Email : asineesh@purdue.edu

Submitted: March 6, 2023

1 Experiment

In this homework, we do the following:

1. Make our own dataset containing around 4000 training and 2000 validation images for bus, cat and pizza classes from the MS-COCO dataset.

2. To train two different models for simultaneous object classification and localization on the above dataset and report their validation accuracy for classification and mean IOU for the bounding box.

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 which contain an object belonging to one of the above three classes with an area of at least 40,000 pixels. This comes to about 4000 training and 2000 validation images and they are resized to (256×256) and saved along with the bounding box coordinates (top left and 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 inspired by ResNet18. It contains 8 ResBlocks with each of them having two convolutional layers. For both the cases, the optimizer used was Adam, the learning rate was 0.001 and the number of epochs trained was 20.

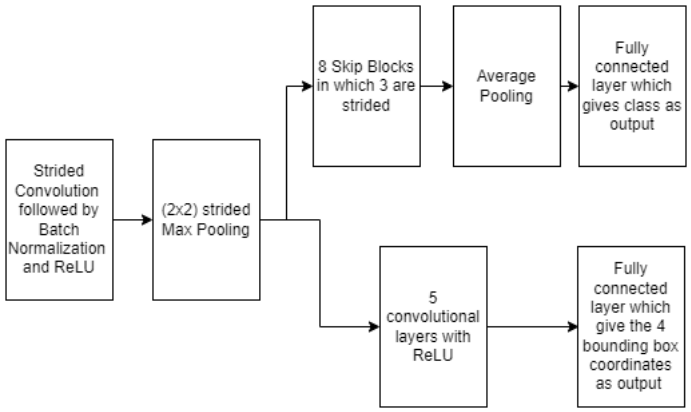


Figure 1: The block diagram for the model. The top half is for classification and the bottom half is for bounding box regression

1.2.1 Task 1

For the first task, the loss used for the classification part was Cross Entropy loss and for the bounding box regression it was MSE loss.

1.2.2 Task 2

For the first task, the loss used for the classification part was Cross Entropy loss and for the bounding box regression it was Complete IOU loss.

2 Results

2.1 Dataset

The following are three images from each of the three classes along with the annotation of the dominant object.

Figure 2: The first row contains images from the cat class, the second column contains images from the pizza class and the third column contains images from the bus class

2.2 Model Training

Below, I have plotted the confusion matrices of the models for validation.

2.2.1 For MSE loss

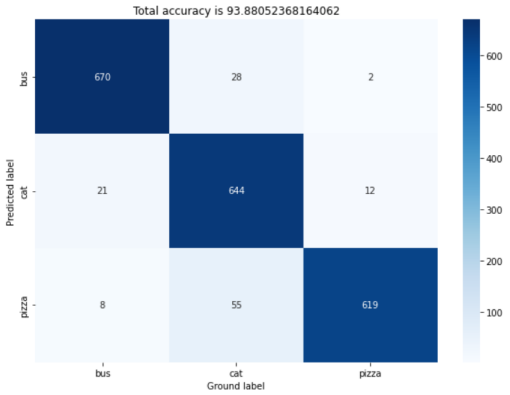


Figure 3: The model trained with MSE for the bounding box regression. The mean IOU is 0.5989

2.2.2 For Complete IOU loss

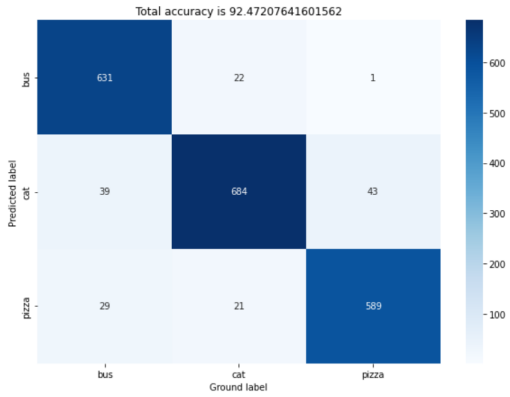
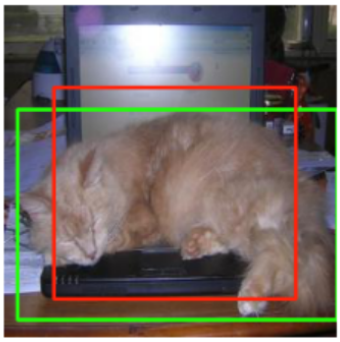


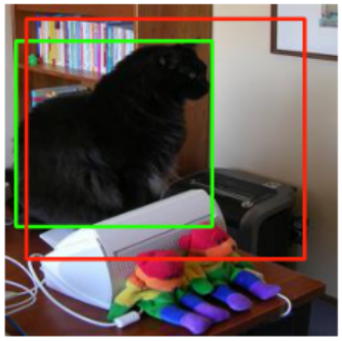
Figure 4: The model trained with Complete IOU for the bounding box regression. The mean IOU is 0.6152

Below are three images from each of the three classes along with their ground truth bounding box in green and predicted bounding box in red.

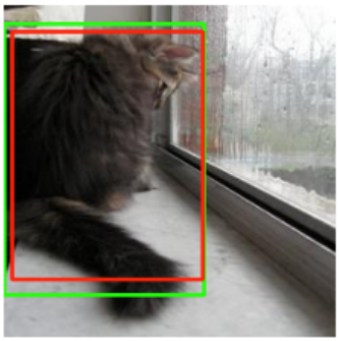
The Ground truth class is cat
The predicted class is cat



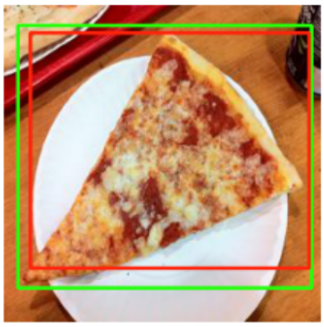
The Ground truth class is cat
The predicted class is cat



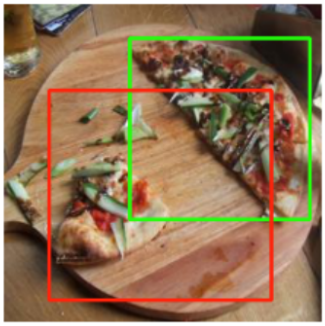
The Ground truth class is cat
The predicted class is cat



The Ground truth class is pizza
The predicted class is pizza

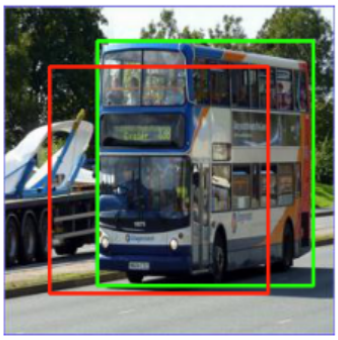


The Ground truth class is pizza
The predicted class is pizza



The Ground truth class is pizza
The predicted class is pizza

The Ground truth class is bus
The predicted class is bus



The Ground truth class is bus
The predicted class is bus

The Ground truth class is bus
The predicted class is bus

3 Conclusion

Overall the classification accuracy of the pizza detector was pretty good (93.88% and 92.47%). The mean IOU score for the model whose bounding box regression part was trained by Complete IOU was slightly better than the one trained by MSE loss (0.6152 vs0.5989).

The results can be further improved by applying more transforms to the images during training, increased regularization etc. In the center image and the bottom right image presented above, you can see that the predicted bounding box includes the object of the same class which is not dominant as well. Thus, the images in the dataset can contain just one object in them to improve the bounding box IOU.

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

    dictionary = {} #stores the annotations
    for catId in catIds:
        category = class_list[coco_labels_inverse[catId]]
        print(category)   #Prints the category
        imgIds = coco.getImgIds(catIds=catId ) #Getting all the ids of the images of the
        total_images=0
        print("The total number of images is "+str(len(imgIds)))
        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'],catIds=catId,iscrowd=False) #Gets the
            anns = coco.loadAnns(annIds) #gets the actual annotations

            for k in range(len(anns)):
                if(anns[k]["area"]>40000 and anns[k]["category_id"]==catId): #Checking i
                    #Resizing the bounding box coordintaes
                    [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]

                    if total_images==0: #Creating the directories
                        dataset = os.path.join("Dataset",dtype,category)
                        os.makedirs(dataset)
                    im1.save(os.path.join(dataset,str(total_images)+".jpeg")) #Saving th
                    dictionary[os.path.join(dataset,str(total_images)+".jpeg")] = [x,y,w

                    total_images = total_images+1
                    break

                else:
                    continue
        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=27.74s)
creating index...
index created!
loading annotations into memory...
Done (t=14.28s)
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
```

```
bus
The total number of images is 2791
The total number of images with a dominant object is 1436
cat
The total number of images is 2818
The total number of images with a dominant object is 1277
pizza
The total number of images is 2202
The total number of images with a dominant object is 1240
```

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
```

```
bus
The total number of images is 1350
The total number of images with a dominant object is 699
cat
The total number of images is 1480
The total number of images with a dominant object is 727
pizza
The total number of images is 1117
The total number of images with a dominant object is 633
```

In [6]:

```
#Number of images in the training dataset
len(train_dict)
```

Out[6]:

```
3953
```

In [7]:

```
#Number of images in the validation dataset
len(test_dict)
```

Out[7]:

```
2059
```

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
import seaborn as sn
```

In [2]:

```
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('/content/drive/MyDrive/Dataset_HW5/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('/content/drive/MyDrive/Dataset_HW5/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 __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(os.path.dirname(img_name)) + "\\" + os.pa
        bbox = torch.Tensor(self.bbox_data[bbox_loc])/256.0
        bbox[2] = bbox[0]+bbox[2]
        bbox[3] = bbox[1]+bbox[3]
        return image, self.labels_map[os.path.basename(os.path.dirname(img_name))], bbox
```

In [3]:

```
#Function to plot the image with its ground truth bounding box
def plot_image(dataset,index):
  img = dataset[index][0] #Getting the image
  [x1,y1,x2,y2] = dataset[index][2]*256.0 #Getting the bounding box coordinates
  i = np.transpose(np.asarray(img*127.5 + 127.5).astype(int),(1,2,0)) #converting the im
  ci = np.ascontiguousarray(i, dtype=np.uint8) #Making the array contiguous
  ri = cv2.rectangle(ci,(int(x1),int(y1)),(int(x2),int(y2)),(36,255,12),2) #drawing the
  plt.imshow(ri)  #plotting the image with the bounding box
  print(dataset[index][1])
```

In [4]:

```
#Function to train the model
#Inspired from the HW4 Question
def training(epochs,optimizer,criterion1,criterion2,net,train_data_loader,device):
  rtrain_losses = []
  ctrain_losses = []
  for epoch in range(epochs):
    running_lossc = 0.0
    running_lossr = 0.0
    for i, data in enumerate(train_data_loader):
      inputs, labels,bbox = data
      inputs = inputs.to(device)
      labels = labels.to(device)
      bbox = bbox.to(device)
      optimizer.zero_grad()
      outputs1,outputs2 = net(inputs)
      loss1 = criterion1(outputs1, labels)
      loss2 = criterion2(outputs2, bbox)
      loss1.backward(retain_graph=True)
      loss2.backward()
      optimizer.step()
      running_lossc += loss1.cpu().item()
      running_lossr += loss2.cpu().item()
      if (i + 1) % 100 == 0:
        print("[epoch: %d, batch: %5d] loss: %.3f" % (epoch + 1, i + 1, (running_lossc+r
    ctrain_losses.append(running_lossc/len(train_data_loader.dataset))
    rtrain_losses.append(running_lossr/len(train_data_loader.dataset))
  return net, ctrain_losses, rtrain_losses
```

In [5]:

```
#Function to compute the confusion matrix and the mean IOU
def confusion_matrix(model,test_data_loader):
  matrix = torch.zeros((3,3))
  total_iou = 0.0
  with torch.no_grad():
    for b, (X_test, y_test,bbox_test) in enumerate(test_data_loader):
      model.eval()
      X_test, y_test, bbox_test = X_test.to(device), y_test.to(device), bbox_test.to(dev
      # Apply the model
      y_val,bbox_val = model(X_test)
      batch_iou = ops.box_iou(bbox_val,bbox_test)
      total_iou += batch_iou.trace()

      # Tally the number of correct predictions
      predicted = torch.max(y_val.data, 1)[1]
      for i in range(len(predicted)):
        matrix[predicted[i].cpu(),y_test[i].cpu()] += 1

  mean_iou = total_iou/len(test_data_loader.dataset)
  heat = pd.DataFrame(matrix, index = [i for i in ["bus","cat","pizza"]],
                  columns = [i for i in ["bus","cat","pizza"]])
  heat = heat.astype(int)
  accuracy = (matrix.trace()/matrix.sum())*100
  plt.figure(figsize = (10,7))
  plt.title("Total accuracy is "+str(accuracy.item()))

  s = sn.heatmap(heat, annot=True,cmap='Blues',fmt='g')
  s.set(xlabel='Ground label', ylabel='Predicted label')
  print("The IOU mean is " + str(mean_iou.item()))
```

In [6]:

```
#ResBlock architecture
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: #if the output has different size or number of channels
      identity = self.conv3(identity)
      identity = self.bn3(identity)

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

    return x
```

In [7]:

```
#Model architecture containing resblocks
#Inspired by ResNet18
class HW5Net(nn.Module):
  def __init__(self):
    super().__init__()
    #Common layers
    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)
    #Layers for classification
    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.avgpool = nn.AdaptiveAvgPool2d((1,1))
    self.fc = nn.Linear(512,3)
    #Layers for regression
    self.convr1  = nn.Conv2d(64,64,kernel_size=3,padding=1)
    self.convr2  = nn.Conv2d(64,64,kernel_size=3,padding=1)
    self.convr3  = nn.Conv2d(64,64,kernel_size=3,padding=1)
    self.convr4  = nn.Conv2d(64,64,kernel_size=3,padding=1)
    self.convr5  = nn.Conv2d(64,64,kernel_size=3,padding=1)
    self.fr1 = nn.Linear(262144,1024)
    self.fr2 = nn.Linear(1024,4)
    self.sig = nn.Sigmoid()

  def forward(self,x):
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)
    #For classification
    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 = self.avgpool(xc)
    xc = xc.view(xc.shape[0],-1)
    xc = self.fc(xc)
    #For regression
    xr = self.relu(self.convr1(x))
    xr = self.relu(self.convr2(xr))
    xr = self.relu(self.convr3(xr))
    xr = self.relu(self.convr4(xr))
    xr = self.relu(self.convr5(xr))
    xr = xr.view(xr.shape[0],-1)
    xr = self.relu(self.fr1(xr))
    xr = self.sig(self.fr2(xr))
    return xc,xr
```

In [8]:

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

Out[8]:

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

In [9]:

```
train_dataset = IndexedDataset("/content/drive/MyDrive/Dataset_HW5/train")
test_dataset = IndexedDataset("/content/drive/MyDrive/Dataset_HW5/test")
train_dataloader = DataLoader(train_dataset,batch_size=16,shuffle=True,num_workers=16)
test_dataloader = DataLoader(test_dataset,batch_size=16,shuffle=True,num_workers=16)
print(len(train_dataset))
print(len(test_dataset))
```

```
3953
2059

/usr/local/lib/python3.8/dist-packages/torch/utils/data/dataloader.py:55
4: UserWarning: This DataLoader will create 16 worker processes in total.
Our suggested max number of worker in current system is 2, which is small
er than what this DataLoader is going to create. Please be aware that exc
essive 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 [10]:

```
#Function to compute complete IOU loss
def IOUloss(a,b):
  ans = ops.complete_box_iou(a,b)
  return 1*a.shape[0]-ans.trace()
```

In [11]:

```
model1 = HW5Net()
model1 = model1.to(device)
optimizer1 = torch.optim.Adam(model1.parameters(), lr=0.0001, betas = (0.9,0.99))
criterion1 = nn.CrossEntropyLoss()
criterion2 = nn.MSELoss()

model2 = HW5Net()
model2 = model2.to(device)
optimizer2 = torch.optim.Adam(model2.parameters(), lr=0.0001, betas = (0.9,0.99))
criterion3 = nn.CrossEntropyLoss()
criterion4 = IOUloss
epochs = 20
```

In [12]:

```
#Number of learnable layers in the model
len(list(model1.parameters()))
```

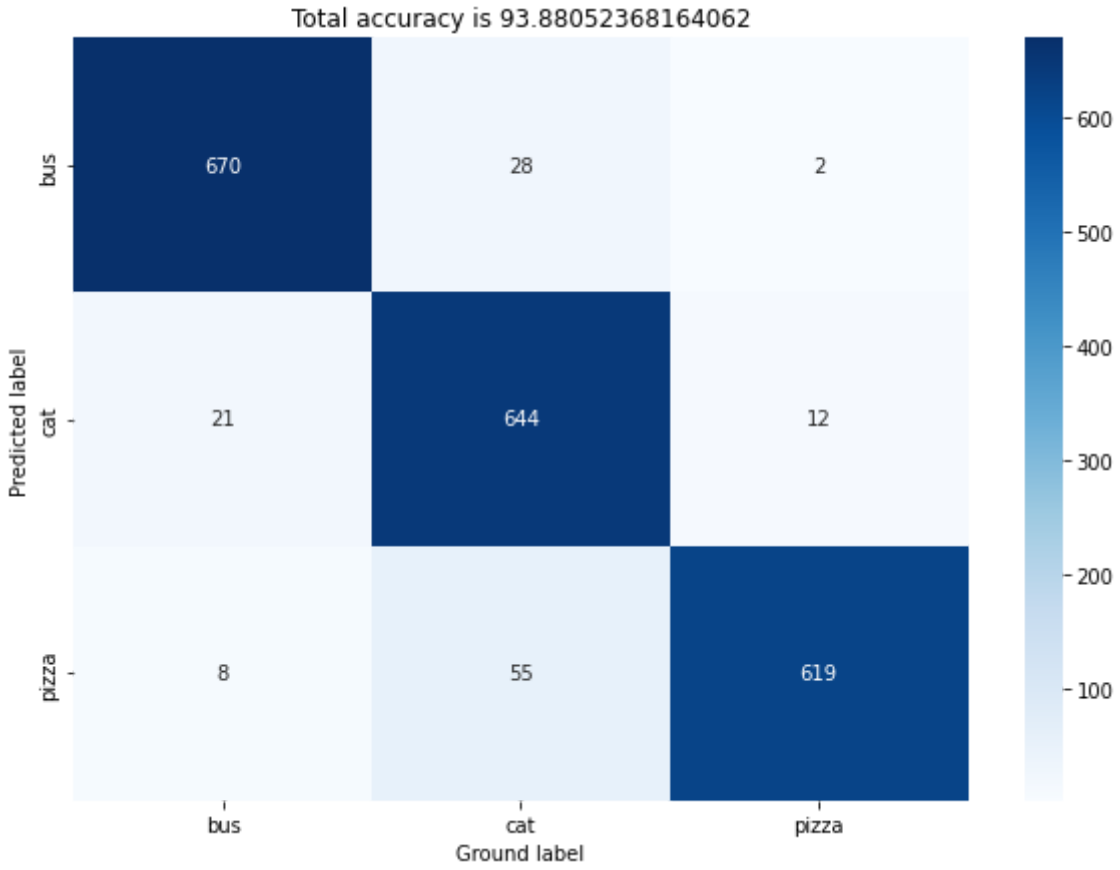
Out[12]:

116

In [14]:

```
trained_model1,ctrain_losses1, rtrain_losses1 = training(epochs,optimizer1,criterion1,cr
confusion_matrix(trained_model1,test_dataloader)
```

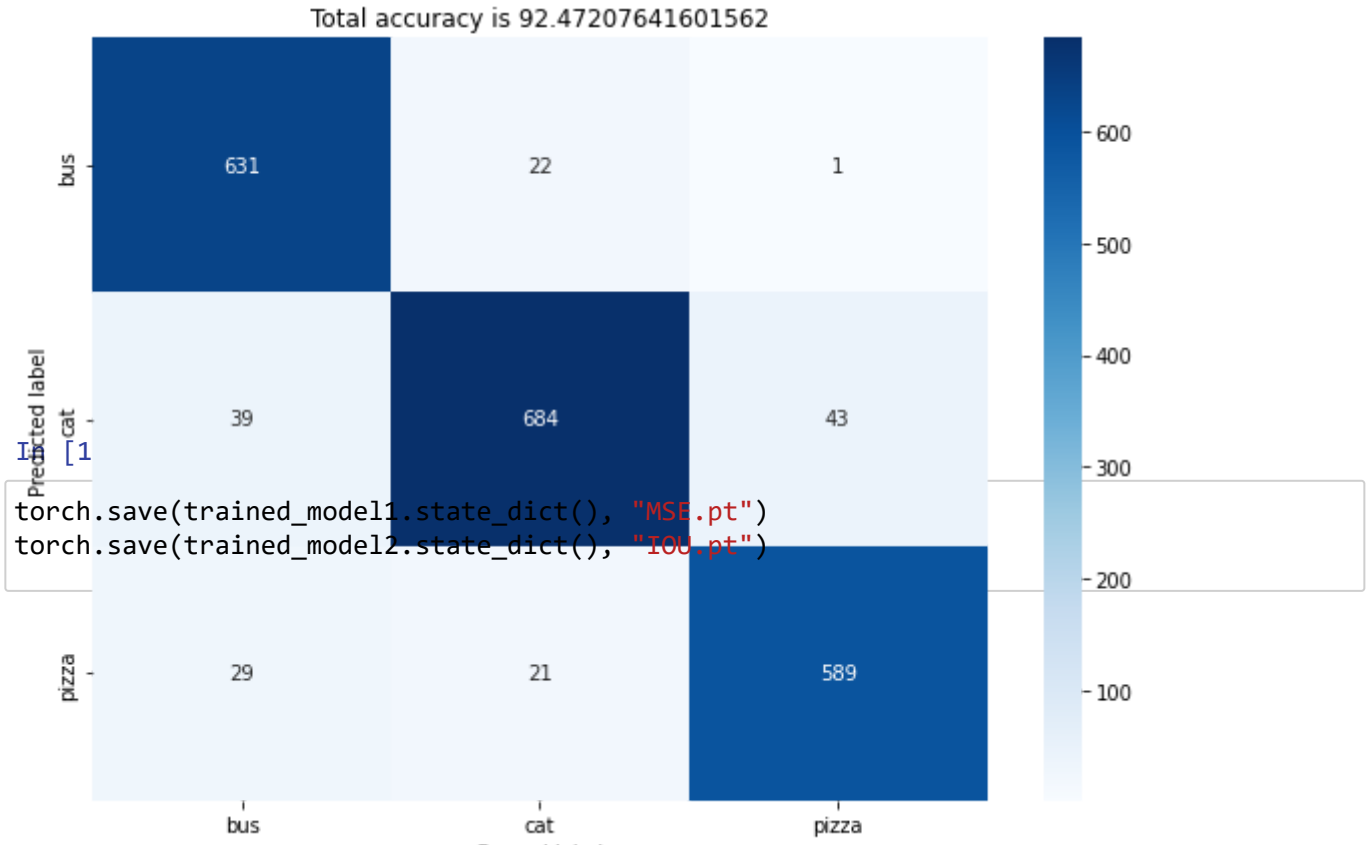
```
[epoch: 1, batch:   100] loss: 0.621
[epoch: 1, batch:   200] loss: 0.507
[epoch: 2, batch:   100] loss: 0.345
[epoch: 2, batch:   200] loss: 0.327
[epoch: 3, batch:   100] loss: 0.305
[epoch: 3, batch:   200] loss: 0.292
[epoch: 4, batch:   100] loss: 0.225
[epoch: 4, batch:   200] loss: 0.249
[epoch: 5, batch:   100] loss: 0.202
[epoch: 5, batch:   200] loss: 0.216
[epoch: 6, batch:   100] loss: 0.185
[epoch: 6, batch:   200] loss: 0.178
[epoch: 7, batch:   100] loss: 0.148
[epoch: 7, batch:   200] loss: 0.150
[epoch: 8, batch:   100] loss: 0.105
[epoch: 8, batch:   200] loss: 0.130
[epoch: 9, batch:   100] loss: 0.094
[epoch: 9, batch:   200] loss: 0.093
[epoch: 10, batch:   100] loss: 0.086
[epoch: 10, batch:   200] loss: 0.097
[epoch: 11, batch:   100] loss: 0.067
[epoch: 11, batch:   200] loss: 0.073
[epoch: 12, batch:   100] loss: 0.075
[epoch: 12, batch:   200] loss: 0.058
[epoch: 13, batch:   100] loss: 0.055
[epoch: 13, batch:   200] loss: 0.055
[epoch: 14, batch:   100] loss: 0.083
[epoch: 14, batch:   200] loss: 0.061
[epoch: 15, batch:   100] loss: 0.057
[epoch: 15, batch:   200] loss: 0.049
[epoch: 16, batch:   100] loss: 0.040
[epoch: 16, batch:   200] loss: 0.040
[epoch: 17, batch:   100] loss: 0.036
[epoch: 17, batch:   200] loss: 0.045
[epoch: 18, batch:   100] loss: 0.016
[epoch: 18, batch:   200] loss: 0.025
[epoch: 19, batch:   100] loss: 0.032
[epoch: 19, batch:   200] loss: 0.046
[epoch: 20, batch:   100] loss: 0.038
[epoch: 20, batch:   200] loss: 0.032
The IOU mean is 0.5989376902580261
```


Total accuracy is 93.88052368164062
bus
cat
pizza
Predicted label
670
28
2
21
644
12
8
55
619
bus
cat
pizza
Ground label
600
500
400
300
200
100

In [13]:

```
trained_model2,ctrain_losses2, rtrain_losses2 = training(epochs,optimizer2,criterion3,IO
confusion_matrix(trained_model2,test_dataloader)
```

```
[epoch: 1, batch:   100] loss: 8.581
[epoch: 1, batch:   200] loss: 8.048
[epoch: 2, batch:   100] loss: 7.134
[epoch: 2, batch:   200] loss: 7.097
[epoch: 3, batch:   100] loss: 6.876
[epoch: 3, batch:   200] loss: 6.811
[epoch: 4, batch:   100] loss: 6.675
[epoch: 4, batch:   200] loss: 6.535
[epoch: 5, batch:   100] loss: 6.322
[epoch: 5, batch:   200] loss: 6.270
[epoch: 6, batch:   100] loss: 5.976
[epoch: 6, batch:   200] loss: 5.966
[epoch: 7, batch:   100] loss: 5.629
[epoch: 7, batch:   200] loss: 5.586
[epoch: 8, batch:   100] loss: 5.354
[epoch: 8, batch:   200] loss: 5.309
[epoch: 9, batch:   100] loss: 4.880
[epoch: 9, batch:   200] loss: 4.906
[epoch: 10, batch:   100] loss: 4.719
[epoch: 10, batch:   200] loss: 4.656
[epoch: 11, batch:   100] loss: 4.442
[epoch: 11, batch:   200] loss: 4.391
[epoch: 12, batch:   100] loss: 3.998
[epoch: 12, batch:   200] loss: 4.090
[epoch: 13, batch:   100] loss: 3.820
[epoch: 13, batch:   200] loss: 3.810
[epoch: 14, batch:   100] loss: 3.533
[epoch: 14, batch:   200] loss: 3.543
[epoch: 15, batch:   100] loss: 3.485
[epoch: 15, batch:   200] loss: 3.387
[epoch: 16, batch:   100] loss: 3.164
[epoch: 16, batch:   200] loss: 3.226
[epoch: 17, batch:   100] loss: 3.032
[epoch: 17, batch:   200] loss: 3.079
[epoch: 18, batch:   100] loss: 3.000
[epoch: 18, batch:   200] loss: 2.933
[epoch: 19, batch:   100] loss: 2.678
[epoch: 19, batch:   200] loss: 2.696
[epoch: 20, batch:   100] loss: 2.638
[epoch: 20, batch:   200] loss: 2.634
The IOU mean is 0.615219235420227
```



In [1

```
torch.save(trained_model1.state_dict(), "MSE.pt")
torch.save(trained_model2.state_dict(), "IOU.pt")
```

In [ ]:

```
#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 [ ]:

```
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('/content/drive/MyDrive/Dataset_HW5/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('/content/drive/MyDrive/Dataset_HW5/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 __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(os.path.dirname(img_name)) + "\\" + os.pa
        bbox = torch.Tensor(self.bbox_data[bbox_loc])/256.0
        bbox[2] = bbox[0]+bbox[2]
        bbox[3] = bbox[1]+bbox[3]
        return image, self.labels_map[os.path.basename(os.path.dirname(img_name))], bbox
```

In [ ]:

```
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 [ ]:

```
class HW5Net(nn.Module):
  def __init__(self):
    super().__init__()
    #Common layers
    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)
    #Layers for classification
    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.avgpool = nn.AdaptiveAvgPool2d((1,1))
    self.fc = nn.Linear(512,3)
    #Layers for regression
    self.convr1  = nn.Conv2d(64,64,kernel_size=3,padding=1)
    self.convr2  = nn.Conv2d(64,64,kernel_size=3,padding=1)
    self.convr3  = nn.Conv2d(64,64,kernel_size=3,padding=1)
    self.convr4  = nn.Conv2d(64,64,kernel_size=3,padding=1)
    self.convr5  = nn.Conv2d(64,64,kernel_size=3,padding=1)
    self.fr1 = nn.Linear(262144,1024)
    self.fr2 = nn.Linear(1024,4)
    self.sig = nn.Sigmoid()

  def forward(self,x):
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)
    #For classification
    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 = self.avgpool(xc)
    xc = xc.view(xc.shape[0],-1)
    xc = self.fc(xc)
    #For regression
    xr = self.relu(self.convr1(x))
    xr = self.relu(self.convr2(xr))
    xr = self.relu(self.convr3(xr))
    xr = self.relu(self.convr4(xr))
    xr = self.relu(self.convr5(xr))
    xr = xr.view(xr.shape[0],-1)
    xr = self.relu(self.fr1(xr))
    xr = self.sig(self.fr2(xr))
```

```
    return xc,xr
```

In [72]:

```
#Function to Plot the image with the ground truth and predicted annotations
def plot_image(dataset,index,model,device):
  img = dataset[index][0] #Getting the image
  [x1,y1,x2,y2] = dataset[index][2]*256.0 #Getting the ground truth bounding box coordir
  i = np.transpose(np.asarray(img*127.5 + 127.5).astype(int),(1,2,0)) #converting the im
  ci = np.ascontiguousarray(i, dtype=np.uint8) #Making the array contiguous
  ri = cv2.rectangle(ci,(int(x1),int(y1)),(int(x2),int(y2)),(36,255,12),2) #drawing the

  model.eval()
  bimg = img.unsqueeze(dim=0).to(device) #reshaping the image to pass through the model
  a,b = model(bimg)
  pred = torch.max(a.data, 1)[1].item() #Getting the predicted class
  bbox = np.asarray(b.cpu().detach().numpy().squeeze())*256 #Getting the predicted bound
  fi = cv2.rectangle(ri,(int(bbox[0]),int(bbox[1])),(int(bbox[2]),int(bbox[3])),(255,36,

  plt.imshow(fi)  #plotting the image with the bounding box
  plt.axis("off")
  labels_map = {0:"bus", 1: "cat", 2: "pizza"}
  print("The Ground truth class is "+ labels_map[dataset[index][1]]) #Printing the grour
  print("The Predicted class is " +labels_map[pred]) #Printing the predicted class
```

In [75]:

```
#Function to Plot the image with the ground truth annotation
def plot_image_ground(dataset,index):
  img = dataset[index][0] #Getting the image
  labels_map = {0:"bus", 1: "cat", 2: "pizza"}
  cl = labels_map[dataset[index][1]] #Class of the image
  [x1,y1,x2,y2] = dataset[index][2]*256.0 #Getting the ground truth bounding box coordir
  i = np.transpose(np.asarray(img*127.5 + 127.5).astype(int),(1,2,0)) #converting the im
  ci = np.ascontiguousarray(i, dtype=np.uint8) #Making the array contiguous
  ri = cv2.rectangle(ci,(int(x1),int(y1)),(int(x2),int(y2)),(36,255,12),2) #drawing the
  fi = cv2.putText(ri,cl,(int(x1),int(y1-10)),cv2.FONT_HERSHEY_SIMPLEX,0.8,(36,255,12),2
  return fi
```

In [ ]:

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

Out[6]:

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

In [8]:

```
#Loading the models
model1 = HW5Net().to(device)
model2 = HW5Net().to(device)
model1.load_state_dict(torch.load("/content/drive/MyDrive/HW5_models/MSE.pt"))
model2.load_state_dict(torch.load("/content/drive/MyDrive/HW5_models/IOU.pt"))
```

In [13]:

```
#loading the dataset
test_dataset = IndexedDataset("/content/drive/MyDrive/Dataset_HW5/test")
```

In [73]:

```
#Plotting an image along with ground truth and predicted annotations
plot_image(test_dataset,1828,model1,device)
```

```
The Ground truth class is bus
The Predicted class is bus
```

In [81]:

```
#Plotting the 3 images each of the 3 classes along with ground truth annotation
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(test_dataset,4))
ax12.imshow(plot_image_ground(test_dataset,2))
ax13.imshow(plot_image_ground(test_dataset,3))

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(test_dataset,801))
ax22.imshow(plot_image_ground(test_dataset,800))
ax23.imshow(plot_image_ground(test_dataset,803))

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(test_dataset,1899))
ax32.imshow(plot_image_ground(test_dataset,1902))
ax33.imshow(plot_image_ground(test_dataset,1903))
```

Out[81]:

```
<matplotlib.image.AxesImage at 0x7f71ca2f12e0>
```


cat
cat
cat
pizza
pizza
pizza
bus
bus
bus