SPRING 2023 ECE 60146 – Homework 5

Sahithi Kodali – 34789866
kodali1@purdue.edu

RESULTS:

3.1 Creating own Object Localization dataset:

A dataset with a total of 3954 training images and 2059 validation images has been created which includes at least one object from the categories [‘bus’, ‘cat’, ‘pizza’]. The bounding box criteria of 40000 pixels area and resizing to 256x256 size images has been applied to attain this custom dataset.

A sample of three images from each of the class with its scaled bounding box have been plotted as shown in Figure 1.

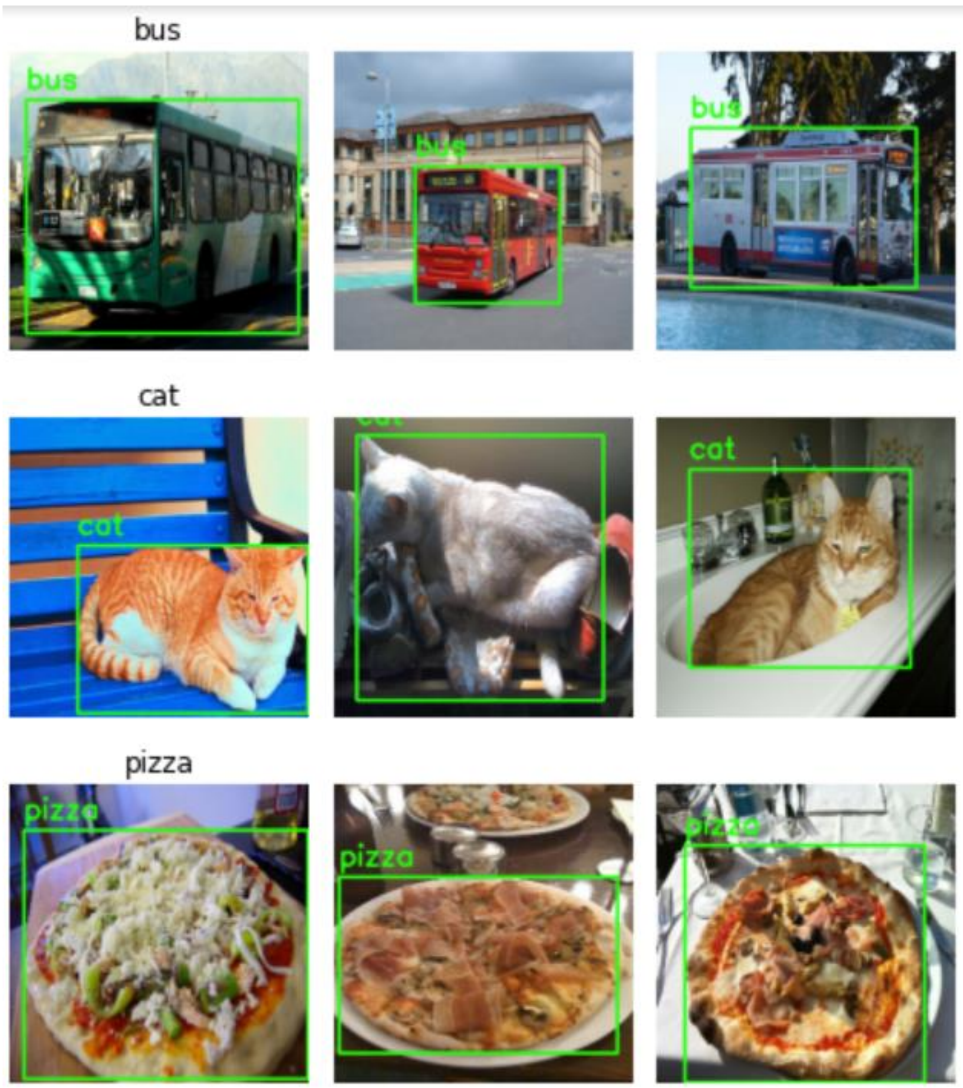

Figure 1: Plot of three images from each class with a boundary box detecting the object

4.2 Building Deep Neural Network

To implement a deep CNN for simultaneous object detection and location, a Skip-connection block/ResBlock is used inside the network with the *total learnable layers in the model being 70*. The snippet of the network using ResBlock can be seen in Figure 2.

```python
#ResNet Block (Inspired from DLStudio SkipBlock)

class ResBlock(nn.Module):
  def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
    super(ResBlock, self).__init__()
    self.downsample = downsample
    self.skip_connections = skip_connections
    self.in_ch = in_ch
    self.out_ch = out_ch

    #convolution layer adn batch normalization
    self.conv1 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
    norm_layer1 = nn.BatchNorm2d
    self.bn1 = norm_layer1(out_ch)

    #downsampler - convolution layer
    if downsample:
        self.downsampler = nn.Conv2d(in_ch, out_ch, 1, stride=2)

  def forward(self, x):

    #residual input
    identity = x

    #convolution adn batch normalization, relu
    out = self.conv1(x)
    out = self.bn1(out)
    out = torch.nn.functional.relu(out)

    #check for input and output channels
    if self.in_ch == self.out_ch:
        out = torch.nn.functional.relu(out)

    #check downsampling
    if self.downsample:
        out = self.downsampler(out)
        identity = self.downsampler(identity)

    #check for skip connections
    if self.skip_connections:
        if self.in_ch == self.out_ch:
            out = out + identity
        else:
            ## To understand the following assignments, recall that the data has the
            ## shape [B,C,H,W]. So it is the second axis that corresponds to the channels
            out[:,:self.in_ch,:,:] = out[:,:self.in_ch,:,:] + identity
            out[:,self.in_ch:,:,:] = out[:,self.in_ch:,:,:] + identity
    return out
```

[111]

```
#Detection and Localization Network (Inspired from DL Studio and HW2-2022)

class DetectAndLocalize(nn.Module):
 def __init__(self,skip_connection = True):
  super(DetectAndLocalize, self).__init__()
  self.skip_connection = skip_connection

  if self.skip_connection:

    #Classification
    #three resblock layers each followed by with downsampling
    self.skip_block_c1 = ResBlock(64, 64)
    self.skip_block_c1ds = ResBlock(64, 64, downsample=True)

    self.skip_block_c2 = ResBlock(64,64)
    self.skip_block_c2ds = ResBlock(64,64, downsample=True)

    self.skip_block_c3 = ResBlock(64,64)
    self.skip_block_c3ds = ResBlock(64,64, downsample=True)

    #two resblock layers
    self.skip_block_c4= ResBlock(64,64)
    self.skip_block_c5= ResBlock(64,64)

    #Regression
    #Three resBlock layers for
    self.skip_block_r1 = ResBlock(64,64)
    self.skip_block_r2 = ResBlock(64,64)
    self.skip_block_r3 = ResBlock(64,64)


  #classification (convolutional layer -> Max pooling -> 2 fully connected layers)
  self.conv = nn.Conv2d(3, 64, 3, padding=1)
  self.pool = nn.MaxPool2d(2, 2)
  self.fc1 = nn.Linear(8 * 8 * 64, 1000)
  self.fc2 = nn.Linear(1000, 3)

  #Regression
  #Sequence of convolution, Batch Normalization and ReLU layers
  self.conv_seqn = nn.Sequential(
      nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(inplace=True),
      nn.Conv2d(in_channels=64, out_channels=64,kernel_size=3,stride=2, padding=1),
      nn.ReLU(inplace=True)
      )

  #Sequnece of linear and ReLU layers
```

```
    self.fc_seqn = nn.Sequential(
        nn.Linear(16384, 2048),
        nn.ReLU(inplace=True),
        nn.Linear(2048, 1024),
        nn.ReLU(inplace=True),
        nn.Linear(1024, 512),
        nn.ReLU(inplace=True),
        nn.Linear(512, 4)
        )

  def forward(self,x):
    x = self.pool(F.relu(self.conv(x)))
    x_c = x.clone()

    #Classsification
    # two skip blocks followed by downsampling blocks
    x_c = self.skip_block_c1(x_c)
    x1 = self.skip_block_c1ds(x_c)

    x2 = self.skip_block_c2(x1)
    x3 = self.skip_block_c2ds(x2)

    #three skip blocks followed by a summation of outputs
    x4 = self.skip_block_c3(x3)
    x5 = self.skip_block_c4(x4)
    x6 = self.skip_block_c5(x5)

    if self.skip_connection:
      x6 = x6 + x5

    #final fully connected layers
    x6 = x6.view(-1, 64 * 8 * 8)
    x6 = F.relu(self.fc1(x6))
    out_cls = self.fc2(x6)

    #Regression
    x_r = x.clone()

    #skip blocks
    x_r1 = self.skip_block_r1(x_r)
    x_r2 = self.skip_block_r2(x_r1)
    x_r3 = self.skip_block_r3(x_r2)

    #convolution and fully connected layers sequence
    r4 = self.conv_seqn(x_r3)
    r4 = r4.view(-1, 64*16*16)
    out_reg = self.fc_seqn(r4)

    return out_cls, out_reg
```

Figure 2: The ResNet block and the network block using ResNet

4.3. Training and Evaluating the Trained Network

An own torch.utils.data.Dataset and DataLoader are implemented based on the own dataset requirements. The network in now trained using the training data and evaluated on validation data. The hyperparameters used for training are 8 epochs, 16 batch size and 2 number of workers.

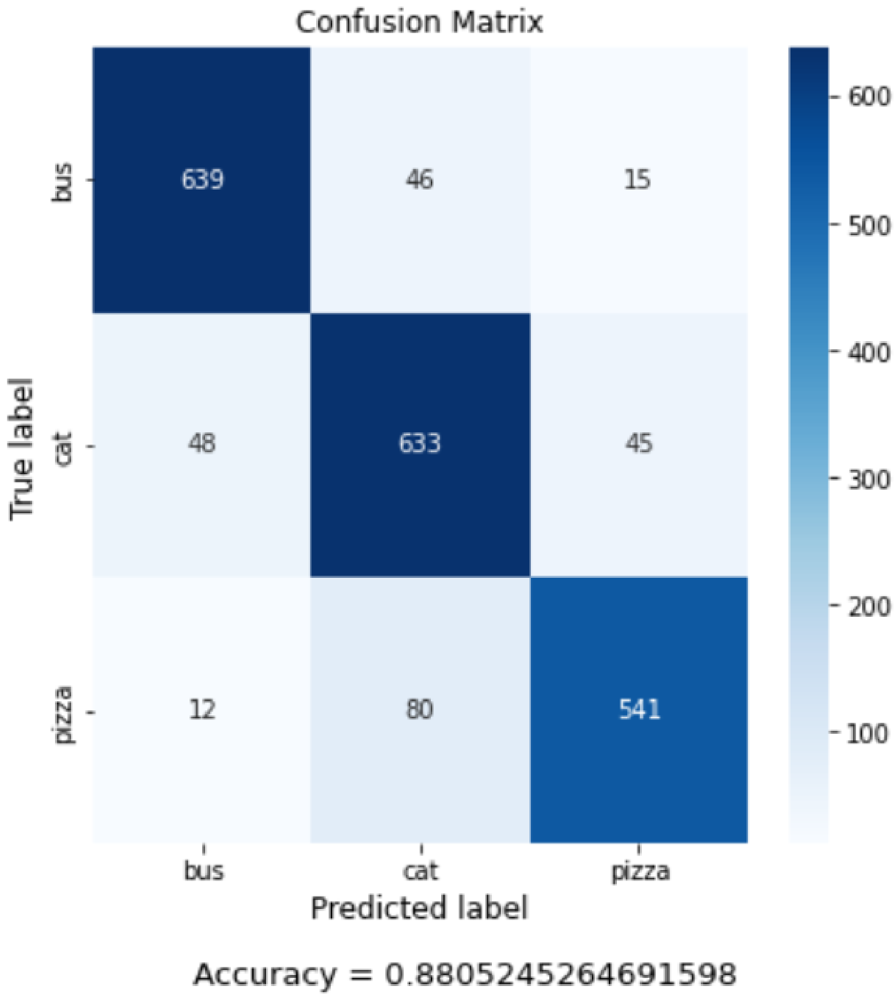

Figure 3: Validation confusion matrix for network with an accuracy of 0.88

The two *mean IoU values attained using MSE loss and Complete IoU loss for regression task are 0.5240, 0.4945* respectively. A plot of 3 images from each class can be seen in Figure 4 with their ground truth and predicted annotation i.e., label and bounding box.

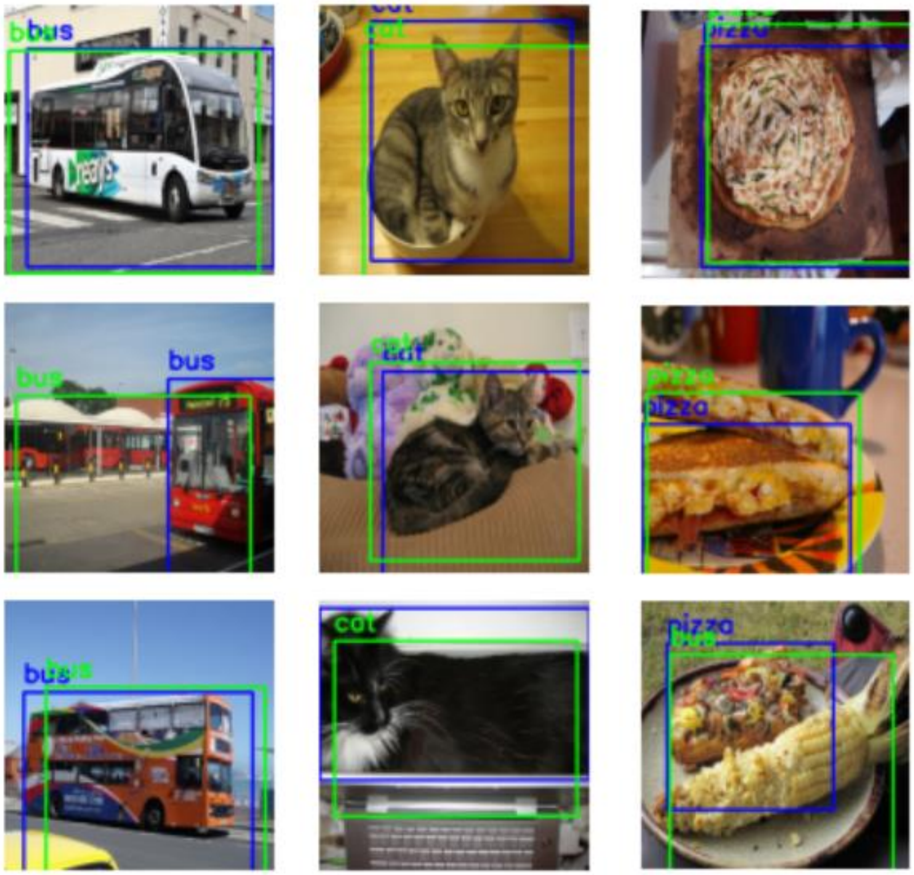

Figure 4: Ground truth (blue) and predicted annotation (green) for 3 images of each class

The performance of the pizza detector is good considering *the accuracy of 88%.* However, this accuracy can still be improved by involving data augmentation of images, along with exploring more combination of the Resblock layers with fully connected layers. It is to be noted that the layers used in the network are to be carefully selected for reducing loss and thus convergence.

SOURCE CODE:

```
# -*- coding: utf-8 -*-
"""HW5_ece60146_new.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1flzgm0MmarKXWP7A7ozKkpvwgkV2kFCr
"""
```

```
# Commented out IPython magic to ensure Python compatibility.
#Change directories, unzip DLStudio and install
# %cd /content/drive/MyDrive/Purdue

# !tar -xzf datasets_for_DLStudio.tar.gz

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/drive/MyDrive/Purdue/DLStudio-2.2.2

!pip install .

# Commented out IPython magic to ensure Python compatibility.
#execute the paying with CIFAR10 code to check the working of the networks
# %cd Examples

pip install pymsgbox

# %run 'object_detection_and_localization.py'

#install COCO API and download data
!pip install pycocotools

# !wget http://images.cocodataset.org/zips/train2014.zip -P
/content/drive/MyDrive/Purdue

# !wget http://images.cocodataset.org/zips/val2014.zip -P
/content/drive/MyDrive/Purdue

# !wget http://images.cocodataset.org/annotations/annotations_trainval2014.zip -P
/content/drive/MyDrive/Purdue

# Commented out IPython magic to ensure Python compatibility.
#import libraries required
# %matplotlib inline
from pycocotools.coco import COCO
import numpy as np
import matplotlib.pyplot as plt
import skimage.io as io
import random
import os
import skimage
from shutil import copyfile
import cv2

classes = ['bus', 'cat', 'pizza']
```

```
bb_area_thresh = 40000

#get the annotation file and call an instance of it
train_annot_path =
'/content/drive/MyDrive/Purdue/annotations_trainval2014/annotations/instances_tra
in2014.json'
train_annot = COCO(train_annot_path)

valid_annot_path =
'/content/drive/MyDrive/Purdue/annotations_trainval2014/annotations/instances_val
2014.json'
valid_annot = COCO(valid_annot_path)

#check the categories for test data
catgs_ids = train_annot.getCatIds(catNms = classes)
catgs = train_annot.loadCats(catgs_ids)
catgs.sort(key=lambda x:x['id'])
print(catgs)

#get image ids for train data
train_ids = []

for c in classes:
  ids = train_annot.getImgIds(catIds = train_annot.getCatIds([c]))
  for i in ids:
    train_ids.append(i)

len(train_ids)

#load test data images
train_imgs_load = train_annot.loadImgs(train_ids)
len(train_imgs_load)

#check the categories for validation data
catgs_ids = valid_annot.getCatIds(catNms = classes)
catgs = valid_annot.loadCats(catgs_ids)
catgs.sort(key=lambda x:x['id'])
print(catgs)

#get image ids for validation data
valid_ids = []

for c in classes:
  ids = valid_annot.getImgIds(catIds = valid_annot.getCatIds([c]))
  for i in ids:
```

```
    valid_ids.append(i)

len(valid_ids)

#load validation data images
valid_imgs_load = valid_annot.loadImgs(valid_ids)
len(valid_imgs_load)

#set labels for categories
train_labels = {}
for i,cls in enumerate(classes):
  for c in catgs:
    if c['name'] == cls:
      train_labels[c['id']] = i

print(train_labels)

#plotting an image to check for correctness of classes (from demo doc)
img_idx = np.random.randint(0,len(train_ids))
img = train_annot.loadImgs(train_ids[img_idx])[0]
I = io.imread(img['coco_url'])

if len(I.shape) == 2:
  I = skimage.color.gray2rgb(I)

plt.axis('off')
plt.imshow(I)
plt.show()

# load and display instance annotations (from demo doc)
annIds = train_annot.getAnnIds(imgIds=img['id'], catIds=catgs_ids, iscrowd=None)
anns = train_annot.loadAnns(annIds)
# train_annot.showAnns(anns)

fig, ax = plt.subplots(1,1)
image = np.uint8(I)

for ann in anns:
  [x, y, w, h] = ann['bbox']
  label = train_labels[ann['category_id']]
  image = cv2.rectangle(image, (int(x), int(y)), (int(x+w), int(y+h)), (36, 255,
12), 2)
  image = cv2.putText(image, classes[label], (int(x), int(y-10)),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (36, 255, 12), 2)
```

```
ax.imshow(image)
ax.set_axis_off()
plt.axis('tight')
plt.show()

#install libraries
from PIL import Image
import urllib.request
import json

#give directories path
train_path = '/content/drive/MyDrive/Purdue/HW5_data/train_data'
valid_path = '/content/drive/MyDrive/Purdue/HW5_data/valid_data'

#Saving training data
for img in train_imgs_load:
  #extract image from source path
  filename, headers = urllib.request.urlretrieve(img['coco_url'])

  #get annotations and filter those that do not belong to our categories
  ann_ids = train_annot.getAnnIds(imgIds=img['id'], catIds=catgs_ids)
  anns = train_annot.loadAnns(ann_ids)
  # print(anns)

  #check for max area
  max_area = 0
  dominant_ann = None
  for ann in anns:
    if ann['area'] > max_area:
      max_area = ann['area']
      dominant_ann = ann

  #if dominant annotaion with area > 40000 exists, resize image, scale bbox
measures
  if dominant_ann is not None and max_area > 40000:
    I = cv2.imread(filename)
    I_resized = cv2.resize(I, (256, 256))

    top_x, top_y, w, h = dominant_ann['bbox']
    top_x_scaled = int(top_x * 256 / I.shape[1])
    top_y_scaled = int(top_y * 256 / I.shape[0])
    w_scaled = int(w * 256 / I.shape[1])
    h_scaled = int(h * 256 / I.shape[0])
    bbox_scaled = [top_x_scaled, top_y_scaled, w_scaled, h_scaled]
```

```
    label_num = train_labels[dominant_ann['category_id']]
    category = classes[label_num]

    #save resized image to train path with its annotations
    cv2.imwrite(os.path.join(train_path, '{}.jpg'.format(dominant_ann['id'])), 
I_resized)
    with open(os.path.join(train_path, '{}.json'.format(dominant_ann['id'])), 
'w') as f:
        json.dump({
            'filename': '{}.jpg'.format(dominant_ann['id']),
            'width': 256,
            'height': 256,
            'category_name': category,
            'label_num' : label_num,
            'bbox': [top_x_scaled, top_y_scaled, w_scaled, h_scaled]
            }, f)

#Saving Validation data
for img in valid_imgs_load:
  #extract image from source path
  filename, headers = urllib.request.urlretrieve(img['coco_url'])

  #get annotations and filter those that do not belong to our categories
  ann_ids = valid_annot.getAnnIds(imgIds=img['id'], catIds=catgs_ids)
  anns = valid_annot.loadAnns(ann_ids)
  # print(anns)

  #check for max area
  max_area = 0
  dominant_ann = None
  for ann in anns:
    if ann['area'] > max_area:
      max_area = ann['area']
      dominant_ann = ann

  #if dominant annotaion with area > 40000 exists, resize image, scale bbox 
measures
  if dominant_ann is not None and max_area > 40000:
    I = cv2.imread(filename)
    I_resized = cv2.resize(I, (256, 256))

    top_x, top_y, w, h = dominant_ann['bbox']
    top_x_scaled = int(top_x * 256 / I.shape[1])
    top_y_scaled = int(top_y * 256 / I.shape[0])
    w_scaled = int(w * 256 / I.shape[1])
```

```
    h_scaled = int(h * 256 / I.shape[0])
    bbox_scaled = [top_x_scaled, top_y_scaled, w_scaled, h_scaled]

    label_num = train_labels[dominant_ann['category_id']]
    category = classes[label_num]

    #save resized image to valid path with its annotations
    cv2.imwrite(os.path.join(valid_path, '{}.jpg'.format(dominant_ann['id'])),
I_resized)
    with open(os.path.join(valid_path, '{}.json'.format(dominant_ann['id'])),
'w') as f:
        json.dump({
            'filename': '{}.jpg'.format(dominant_ann['id']),
            'width': 256,
            'height': 256,
            'category_name': category,
            'label_num' : label_num,
            'bbox': [top_x_scaled, top_y_scaled, w_scaled, h_scaled]
            }, f)

#plot image and check annotations

def plot_imgs(dir_path, cls, imgs, imgs_annots):
  fig, axs = plt.subplots(1, 3, figsize = (6,6))

  for i, img_name in enumerate(imgs):
    I = os.path.join(dir_path, img_name)
    image = Image.open(I)
    image = np.uint8(image)

    annots_path = os.path.join(dir_path, imgs_annots[i])
    with open(annots_path) as f:
      annots = json.load(f)

    [x, y, w, h] = annots['bbox']
    image = cv2.rectangle(image, (int(x), int(y)), (int(x+w), int(y+h)), (36,
255, 12), 2)
    image = cv2.putText(image, annots['category_name'], (int(x), int(y-10)),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (36, 255, 12), 2)

    axs[i].imshow(image)
    axs[i].set_axis_off()
    if i == 0:
      axs[i].set_title(cls)
```

```
  plt.tight_layout()
  fig.savefig('/content/drive/MyDrive/Purdue/cls_{}_imgs.png'.format(cls))

plot_imgs(train_path, 'bus', ['365526.jpg', '365516.jpg', '165131.jpg'],
['365526.json', '365516.json', '165131.json'])
plot_imgs(train_path, 'cat', ['46260.jpg', '46272.jpg', '46289.jpg'],
['46260.json', '46272.json', '46289.json'])
plot_imgs(train_path, 'pizza', ['1070777.jpg', '1070788.jpg', '1071485.jpg'],
['1070777.json', '1070788.json', '1071485.json'])

#import libraries
import torch
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from PIL import Image
import pandas as pd
import torchvision.transforms as tvt
import torch.nn as nn
import torch.nn.functional as F

#check for GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device

#create dataset and dataloader
class MyDataset(Dataset):

    #intializations
    def __init__(self, root_dir, transform = None):
        super().__init__()
        self.root_dir = root_dir
        self.transform = transform
        self.image_paths = []
        self.bbox = []
        self.labels = []
        self.category = []

        #add image paths and corresponding class as a label
        for file in os.listdir(root_dir):
          if file.endswith('.json'):
            with open(os.path.join(self.root_dir, file), 'r') as f:
              img_data = json.load(f)
              self.image_paths.append(os.path.join(self.root_dir,
img_data['filename']))
              self.labels.append(img_data['label_num'])
```

```
            self.category.append(img_data['category_name'])
            self.bbox.append(img_data['bbox'])

    #compute length of dataset
    def __len__(self):
        return len(self.image_paths)

    #downsample images if required to suit model adn task requirements
    @staticmethod
    def downsample(img, b_box):
      img = cv2.resize(img, (64,64), interpolation = cv2.INTER_AREA)
      x1, y1, x2, y2 = b_box
      x_ratio = 64/img.shape[0]
      y_ratio = 64/img.shape[1]
      x1, y1, x2, y2 = x1//x_ratio, y1//y_ratio, x2//x_ratio, y2//y_ratio
      b_box = [x1, y1, x2, y2]
      return img, b_box


    #apply transformations for the image chosen by index
    def __getitem__(self, index):
      img_path = self.image_paths[index]
      bbox = self.bbox[index]
      label = self.labels[index]

      #load image
      img = Image.open(img_path).convert('RGB')
      img = np.array(img)

      #reshape bbox parameters and scale to (0,1) range
      x1, y1, w, h = bbox
      x2 = x1+w
      y2 = y1+h
      new_bbox = [x1, y1, x2, y2]

      img, new_bbox = self.downsample(img, new_bbox)
      img = img/255.
      new_bbox[0] /= img.shape[0]
      new_bbox[1] /= img.shape[1]
      new_bbox[2] /= img.shape[0]
      new_bbox[3] /= img.shape[1]

      #perform transformations if any
      if self.transform:
          img = self.transform(img)
```

```
      new_bbox = torch.tensor(new_bbox, dtype = torch.float32)
      img = torch.tensor(img, dtype = torch.float32)
      label = torch.tensor(label, dtype = torch.float32)
      return img, label, new_bbox, img_path

#initialize the dataset and dataloader and apply transformations as required

transform = tvt.Compose([tvt.ToTensor()])

train_dataset = MyDataset(train_path, transform = transform)
val_dataset = MyDataset(valid_path, transform = transform)

#check for the data length
print(len(train_dataset))
print(len(val_dataset))

import sys

#initialize batch and num workers
batch_size = 16
num_workers = 2

#create dataloader
train_data_loader = DataLoader(train_dataset, batch_size = batch_size, shuffle =
True, num_workers=num_workers)
val_data_loader = DataLoader(val_dataset, batch_size = batch_size, shuffle =
True, num_workers=num_workers)

#ResNet Block (Inspired from DLStudio SkipBlock)

class ResBlock(nn.Module):
  def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
    super(ResBlock, self).__init__()
    self.downsample = downsample
    self.skip_connections = skip_connections
    self.in_ch = in_ch
    self.out_ch = out_ch

    #convolution layer adn batch normalization
    self.conv1 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
    norm_layer1 = nn.BatchNorm2d
    self.bn1 = norm_layer1(out_ch)

    #downsampler - convolution layer
```

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

  def forward(self, x):

    #residual input
    identity = x

    #convolution adn batch normalization,
relu
    out = self.conv1(x)
    out = self.bn1(out)
    out = torch.nn.functional.relu(out)

    #check for input and output channels
    if self.in_ch == self.out_ch:
        out = torch.nn.functional.relu(out)

    #check downsampling
    if self.downsample:
        out = self.downsampler(out)
        identity = self.downsampler(identity)

    #check for skip connections
    if self.skip_connections:
        if self.in_ch == self.out_ch:
            out = out + identity
        else:
            ## To understand the following assignments, recall that the data has
the
            ## shape [B,C,H,W]. So it is the second axis that corresponds to the
channels
            out[:,:self.in_ch,:,:] = out[:,:self.in_ch,:,:] + identity
            out[:,self.in_ch:,:,:] = out[:,self.in_ch:,:,:] +
identity
    return out

#Detection and Localization Network (Inspired from DL Studio and HW2-2022)

class DetectAndLocalize(nn.Module):
 def __init__(self,skip_connection = True):
  super(DetectAndLocalize, self).__init__()
  self.skip_connection = skip_connection

  if self.skip_connection:
```

```
    #Classification
    #three resblock layers each followed by with downsampling
    self.skip_block_c1 = ResBlock(64, 64)
    self.skip_block_c1ds = ResBlock(64, 64, downsample=True)

    self.skip_block_c2 = ResBlock(64,64)
    self.skip_block_c2ds = ResBlock(64,64, downsample=True)

    self.skip_block_c3 = ResBlock(64,64)
    self.skip_block_c3ds = ResBlock(64,64, downsample=True)

    #two resblock layers
    self.skip_block_c4= ResBlock(64,64)
    self.skip_block_c5= ResBlock(64,64)

    #Regression
    #Three resBlock layers for
    self.skip_block_r1 = ResBlock(64,64)
    self.skip_block_r2 = ResBlock(64,64)
    self.skip_block_r3 = ResBlock(64,64)


  #classification (convolutional layer -> Max pooling -> 2 fully connected
layers)
  self.conv = nn.Conv2d(3, 64, 3, padding=1)
  self.pool = nn.MaxPool2d(2, 2)
  self.fc1 = nn.Linear(8 * 8 * 64, 1000)
  self.fc2 = nn.Linear(1000, 3)

  #Regression
  #Sequence of convolution, Batch Normalization and ReLU layers
  self.conv_seqn = nn.Sequential(
      nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
      nn.BatchNorm2d(64),
      nn.ReLU(inplace=True),
      nn.Conv2d(in_channels=64, out_channels=64,kernel_size=3,stride=2,
padding=1),
      nn.ReLU(inplace=True)
      )

  #Sequnece of linear and ReLU layers
  self.fc_seqn = nn.Sequential(
      nn.Linear(16384, 2048),
      nn.ReLU(inplace=True),
```

```
    nn.Linear(2048, 1024),
    nn.ReLU(inplace=True),
    nn.Linear(1024, 512),
    nn.ReLU(inplace=True),
    nn.Linear(512, 4)
    )

def forward(self,x):
  x = self.pool(F.relu(self.conv(x)))
  x_c = x.clone()

  #Classsification
  # two skip blocks followed by downsampling blocks
  x_c = self.skip_block_c1(x_c)
  x1 = self.skip_block_c1ds(x_c)

  x2 = self.skip_block_c2(x1)
  x3 = self.skip_block_c2ds(x2)

  #three skip blocks followed by a summation of outputs
  x4 = self.skip_block_c3(x3)
  x5 = self.skip_block_c4(x4)
  x6 = self.skip_block_c5(x5)

  if self.skip_connection:
    x6 = x6 + x5

  #final fully connected layers
  x6 = x6.view(-1, 64 * 8 * 8)
  x6 = F.relu(self.fc1(x6))
  out_cls = self.fc2(x6)

  #Regression
  x_r = x.clone()

  #skip blocks
  x_r1 = self.skip_block_r1(x_r)
  x_r2 = self.skip_block_r2(x_r1)
  x_r3 = self.skip_block_r3(x_r2)

  #convolution and fully connected layers sequence
  r4 = self.conv_seqn(x_r3)
  r4 = r4.view(-1, 64*16*16)
  out_reg = self.fc_seqn(r4)
```

```
  return out_cls, out_reg

#initialize model
torch.cuda.empty_cache()
model = DetectAndLocalize(skip_connection=True)
model = model.to(device)

#list total number of layers
num_layers = len(list(model.parameters()))
num_layers

#check model summary
from torchsummary import summary

summary(model,(3, 256, 256))

from torchvision.ops import box_iou, complete_box_iou_loss

epochs = 8
learning_rate = 1e-3

#training
ce_train_loss = []
mse_train_loss = []
iou_train_loss = []
torch.autograd.set_detect_anomaly(True)

#give criterion and optimizer
criterion_ce = torch.nn.CrossEntropyLoss()
criterion_mse = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

model.train()
for epoch in range(epochs):
  running_loss_ce = 0.0
  running_loss_mse = 0.0
  running_loss_iou = 0.0

  for i, data in enumerate(train_data_loader):
    img, label, bbox, im_paths = data
    label = label.type(torch.LongTensor)
    img = img.to(device)
    label = label.to(device)
    bbox = bbox.to(device)
```

```
    optimizer.zero_grad()
    label_pred_prob, bbox_pred = model(img)
    # print(label_pred_prob)
    # print(label)

    ce_loss = criterion_ce(label_pred_prob, label)
    ce_loss.backward(retain_graph = True)

    mse_loss = criterion_mse(bbox_pred, bbox)
    mse_loss.backward()

    # iou_loss = complete_box_iou_loss(bbox_pred, bbox, reduction='mean')
    # iou_loss.backward()

    optimizer.step()

    running_loss_ce += ce_loss.item()
    running_loss_mse += mse_loss.item()
    # running_loss_iou += iou_loss.item()

    #keep track of losses
    if (i+1) % 10 == 0:
      print("[ epoch : %d, batch : %5d] ce_loss : %.3f" %(epoch + 1, i + 1,
running_loss_ce/10))
      print("[ epoch : %d, batch : %5d] mse_loss : %.3f" %(epoch + 1, i + 1,
running_loss_mse/10))
      # print("[ epoch : %d, batch : %5d] mse_loss : %.3f" %(epoch + 1, i + 1,
running_loss_iou/10))

      ce_train_loss.append(running_loss_ce/10)
      mse_train_loss.append(running_loss_mse/10)
      # iou_train_loss.append(running_loss_iou/10)

      running_loss_ce = 0.0
      running_loss_mse = 0.0
      running_loss_iou = 0.0

#plotting the loss vs iterations
plt.plot(mse_train_loss, label = 'MSE')
# plt.plot(iou_train_loss, label = 'IOU')
plt.plot(ce_train_loss, label = 'CE')

plt.title('Training loss vs Iterations')
plt.xlabel('Iteration')
plt.ylabel('Loss')
```

```
plt.legend()
plt.show()

#checking performance on validation dataset

#import libraries
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns

# make predictions on the test set using net1
model.eval()

im_all_paths = []

y_true = []
y_pred = []

bbox_true_plot = []
bbox_pred_plot = []

iou_pred = []


with torch.no_grad():
    for imgs, labels, bbox, im_paths in val_data_loader:
        for i in im_paths:
          im_all_paths.append(i)
        imgs = imgs.to(device)
        labels = labels.to(device)
        bbox = bbox.to(device)

        labels_logits, bbox_pred = model(imgs)

        labels_pred = labels_logits.argmax(dim=1)
        y_true += labels.cpu().numpy().tolist()
        y_pred += labels_pred.cpu().numpy().tolist()


        iou = box_iou(bbox_pred, bbox)
        iou_pred.append(iou.mean())

        bbox_true_plot += bbox.cpu().numpy().tolist()
        bbox_pred_plot += bbox_pred.cpu().numpy().tolist()

#compute mean IoU
```

```
mean_iou = sum(iou_pred)/len(iou_pred)
print('Mean Iou is', mean_iou)

# print(y_true)
# print(y_pred)

# construct the confusion matrix
cm = confusion_matrix(y_true, y_pred)
acc = accuracy_score(y_true, y_pred)

plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot = True, cmap = 'Blues', xticklabels = classes, yticklabels
= classes, fmt = 'g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted label', fontsize = 12)
plt.ylabel('True label', fontsize = 12)
plt.text(1.7, 3.5, 'Accuracy = ' + str(acc), fontsize = 13, ha='center',
va='center')
plt.show()

#plot 3 images from 3 classes with GT and predicted annotation
classes = ['bus', 'cat', 'pizza']

y_true_int =  [int(x) for x in y_true]

#plot image and check annotations
fig, axs = plt.subplots(9, 2, figsize = (15,15))

idxs = [8, 9,13,14, 0,1,2,3,5,]

for num, i in enumerate(idxs):

  I = im_all_paths[i]

  I = Image.open(I)
  image = np.uint8(I)

  for z in range(len(bbox_true_plot[i])):
    bbox_true_plot[i][z] *= 64
  [x, y, w, h] = bbox_true_plot[i]
  image = cv2.rectangle(image, (int(x), int(y)), (int(x+w), int(y+h)), (0, 0,
255), 2)
  image= cv2.putText(image, classes[y_true_int[i]], (int(x), int(y-10)),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2)
```

```
  axs[num,0].imshow(image)
  axs[num,0].set_axis_off()

  for z in range(len(bbox_pred_plot[i])):
    bbox_pred_plot[i][z] *= 64
  [x, y, w, h] = bbox_pred_plot[i]
  image = cv2.rectangle(image, (int(x), int(y)), (int(x+w), int(y+h)), (0, 255,
0), 2)
  image = cv2.putText(image, classes[y_pred[i]], (int(x), int(y-10)),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)

  axs[num,1].imshow(image)
  axs[num,1].set_axis_off()

plt.tight_layout()
plt.show()
```

****