Homework 5 - Deep Learning (ECE 60146)

Souradip Pal
pal43@purdue.edu
PUID 0034772329

March 6, 2023

1 Introduction

In this homework, the programming tasks give an overview of using a deep neural network for objection detection and localization. It introduces the idea of skip connections or Resnet blocks which are widely employed for extracting meaningful convolutional image features for predicting the dominant object in an image and its corresponding bounding box parameters. Moreover, it experiments with a different type of regression loss called Complete IoU loss for better prediction of the bounding box coordinates. In this assignment, a Convolutional Neural Network is created consisting of Skip Connections to learn image features. The networks were trained and validated using a subset of images present in the COCO dataset(2014 version) which were downloaded with the help of **COCO API**, a library for managing the dataset. Finally, the performance of the networks was compared by computing the confusion matrix to understand which images were correctly or incorrectly labeled for each of the classes and plotting the predicted bounding boxes along with the ground truth boxes. Some ideas related to the logic of downloading the COCO dataset, the architecture of SkipBlock, and the Detection and Localization network were taken from the source code of the **DLStudio** module and from the previous year's solutions for completing this assignment.

2 Methodology

Initially, to get familiar with the code for training and testing object detection and localization networks, the scripts *playing_with_skip_connections.py* & *object_detection_and_localization.py* from the **Examples** directory in the **DLStudio** module were run. These scripts illustrated how the neural networks were constructed with different types of skip connections and gave an idea of the performance difference if skip blocks were used or not used. Using a similar approach for the programming tasks, a custom SkipBlock class was created and used in the *HW5Net* network for predicting the dominant object and its bounding boxes. Next, images with only a single dominant object were filtered based on the area of the bounding box to form a subset of the COCO dataset for this assignment. Training and testing routines were created to train and log the losses in periodic checkpoints. For evaluation, a method was created to calculate the confusion matrix to measure the accuracy of the models and how the models performed in classifying each of the classes. Also, the mean IoU loss was calculated to get the accuracy of the prediction of the bounding boxes. The following section gives a detailed description of each of the approaches and the results obtained from the experiments performed on the object detection ad localization models.

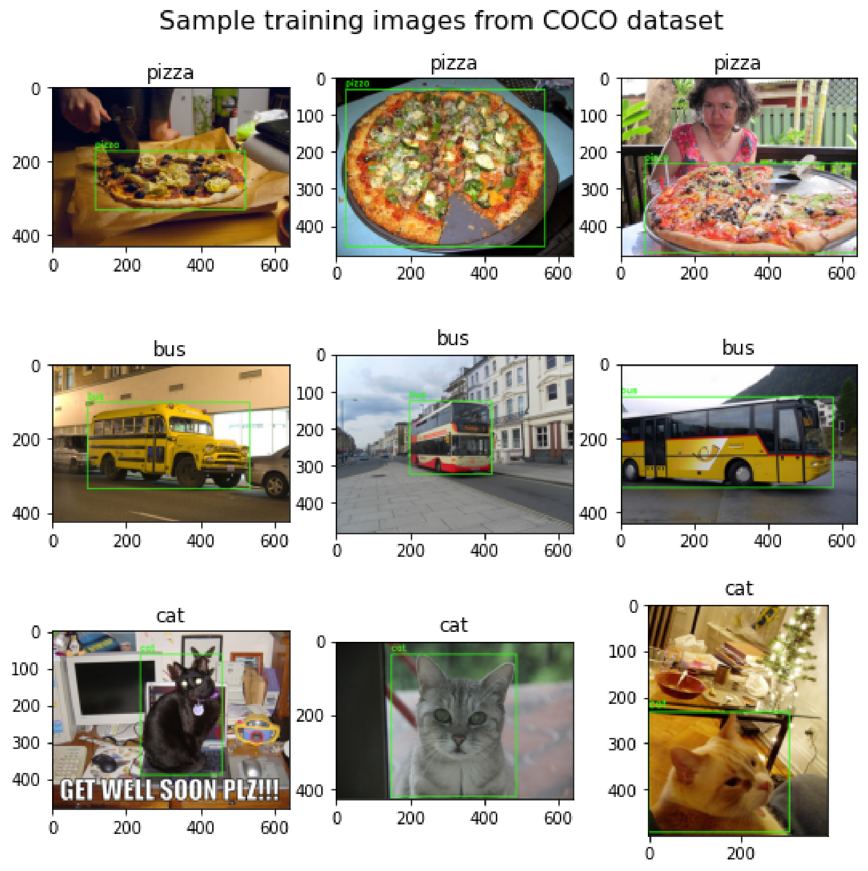


Figure 1: Sample of training images from the COCO dataset

3 Implementation and Results

3.1 Task 3: Programming Tasks

3.1.1 Creating Object Localization Dataset

- In order to use the **COCO API** for managing the MS-COCO dataset, the python version of the API called *pycocotools* was downloaded.
- Since the 2014 version of the COCO dataset was used for this homework, the 2014 Train/Val annotation zip file was downloaded from the MS-COCO website. It contained a JSON file named *instances_train2014.json* which contained details of the images present in the *train2014.zip* and *val2014.zip* files including the image ids, image URLs, image categories, bounding box annotations, etc.
- Here, only the following three categories : [‘pizza’, ‘bus’, ‘cat’] were selected. From these categories, the image ids were filtered based on the area of the dominant object. If an image consists of only a single dominant object and the area of the rectangular bounding box corresponding to that object was greater than 40000 pixels then the image was picked up. The images are then downloaded using those URLs using the *requests* python library and saved in separate directories named after each of the categories. Moreover, the images were downsampled to a smaller size of 256×256 for training and validation. A total of **3779** training images and **2032** testing images

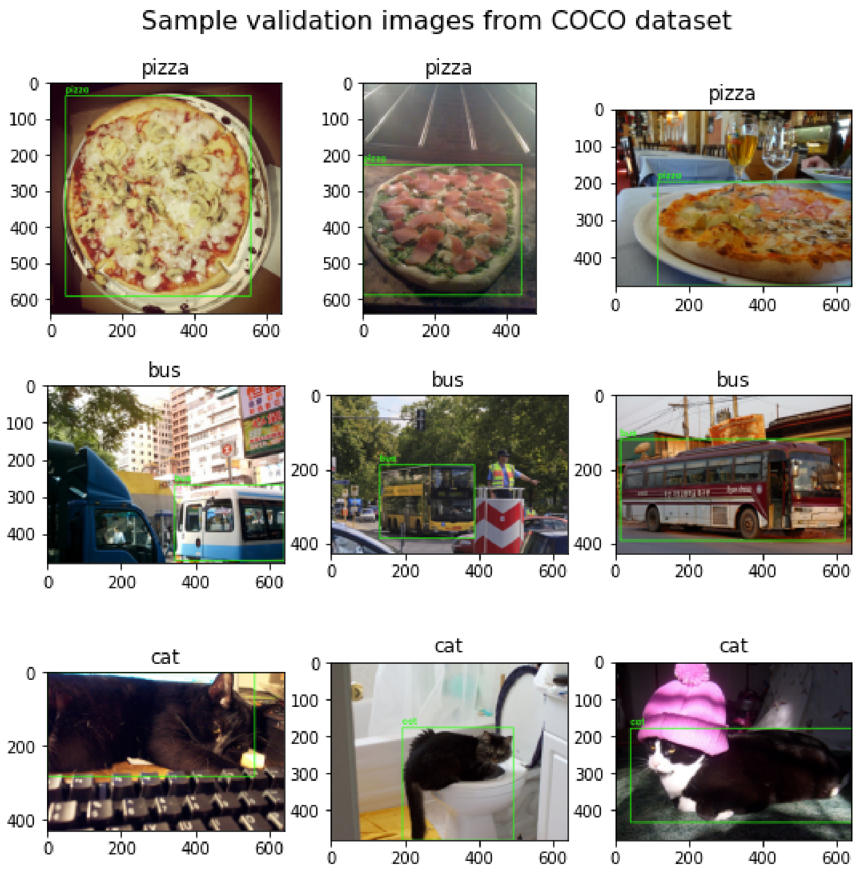


Figure 2: Sample of validation images from the COCO dataset

were obtained after filtering. Shown in Fig. 1 and Fig. 2 are samples of the images from the training and validation sets for each of the three classes with their corresponding ground truth boxes.

- A custom *CocoObjectDetectionDataset* class was created from *torch.utils.data.Dataset* class to load the images from each of the class directories after applying the necessary transforms. The box coordinates for the single-instance object were normalized to a value between $[0, 1]$. Finally, a *Dataloader* was created to wrap the dataset for processing the images in batches of **64** for training and validation.

3.1.2 Building the Deep Neural Network

- **SkipBlock/ResBlock**: In this task, a custom skip block network *ResBlock* was created as per the instructions in the assignment. The ResBlock consists of two convolutional layers followed by BatchNorm layers. In addition to that, LeakyReLU layers($\alpha = 0.01$) were used as activation functions and Dropout layers were introduced. Each of the convolutional layers has a kernel size of 3 and padding of 1 to maintain the output shapes without downsampling. The following code block gives a description of the *ResBlock* network.

```
class ResBlock(nn.Module):
    def __init__(self, in_ch, out_ch, downsample=False):
        super(ResBlock, self).__init__()
        self.downsample = downsample
        self.in_ch = in_ch
        self.out_ch = out_ch
        self.conv1 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(out_ch)
        self.bn2 = nn.BatchNorm2d(out_ch)
        self.dropout = nn.Dropout(0.2)
        self.relu = nn.LeakyReLU()
        if downsample:
            self.downsampler = nn.Conv2d(in_ch, out_ch, 1, stride=2)

    def forward(self, x):
        identity = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        if self.in_ch == self.out_ch:
            out = self.conv2(out)
            out = self.bn2(out)
            out = self.relu(out)
            out = self.dropout(out)

        if self.downsample:
            out = self.downsampler(out)
            identity = self.downsampler(identity)

        if self.in_ch == self.out_ch:
            out = out + identity
        else:
            out[:,:self.in_ch,:,:] += identity
            out[:,self.in_ch:,:,:] += identity
        return out
```

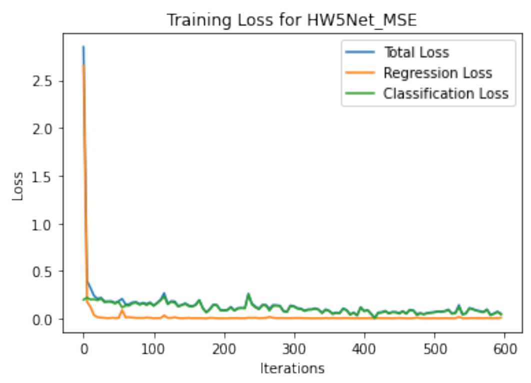
- **HW5Net Architecture**: The HW5Net model was created as per the instruction in the assignment. The previously defined *ResBlock* layers were used sequentially in the network backbone. For the classification and regression network, two separate networks were created called the *class_head* and *box_head* respectively. The *class_head* and *box_head* modules consist of linear layers followed by ReLU layers. The *class_head* block takes in the extracted image features from the backbone and predicts the three class probabilities. The *box_head* block takes in the extracted image features from the backbone and predicts the four values corresponding to the pixel coordinates of the top left and bottom right corners of the bounding box of the single object in each image. The total number of learnable layers of the entire network came out to be **64** and the total number of learnable parameters was $\sim$**86M**. The following code snippet shows the HW5Net network.

```
# Define HW5Net architecture
class HW5Net(nn.Module):
    """ Resnet - based encoder that consists of a few
    downsampling + several Resnet blocks as the backbone
    and two prediction heads .
    """

    def __init__ (self, input_nc, ngf = 8, n_blocks = 4):
        """
        Parameters :
        input_nc (int) -- the number of channels input images
        output_nc (int) -- the number of channels output images
        ngf (int ) -- the number of filters in the first conv layer
        n_blocks (int) -- the number of ResNet blocks
        """
        assert(n_blocks >= 0)
        super(HW5Net, self). __init__ ()
        # The first conv layer
        model = [
            nn.ReflectionPad2d(3),
            nn.Conv2d(input_nc, ngf, kernel_size = 7, padding = 0),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True)
        ]
        # Add downsampling layers
        n_downsampling = 4
        for i in range(n_downsampling):
            mult = 2 ** i
            model += [
                nn.Conv2d(ngf * mult, ngf * mult * 2, kernel_size = 3, stride = 2, padding = 1),
                nn.BatchNorm2d(ngf * mult * 2),
                nn.ReLU(True)
            ]
        # Add your own ResNet blocks
        mult = 2 ** n_downsampling
        for i in range(n_blocks):
            model += [ResBlock(ngf * mult, ngf * mult, downsample = False)]
        self.model = nn.Sequential(*model)
        # The classification head
        class_head = [
            nn.Linear(128*16*16, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.2),
            nn.Linear(512, 64),
            nn.ReLU(inplace=True),
            nn.Linear(64, 3)
        ]
        self.class_head = nn.Sequential(*class_head)
        # The bounding box regression head
        bbox_head = [
            nn.Linear(128*16*16, 2048),
            nn.ReLU(inplace=True),
            nn.Linear(2048, 512),
            nn.ReLU(inplace=True),
            nn.Linear(512, 4)
        ]
        self.bbox_head = nn.Sequential(*bbox_head)

    def forward (self, input):
        ft = self.model(input)
        ft = ft.view(-1, 128*16*16)
        cls = self.class_head(ft)
        bbox = self.bbox_head(ft)
        return cls, bbox
```

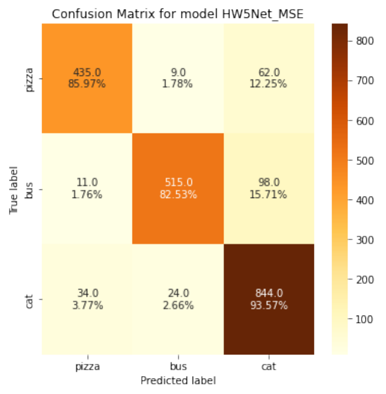
3.1.3 Training and Evaluating the Deep Neural Network

- **MSE Loss** - Using the training routine created, the network was trained for 20 epochs with regression loss as the Mean Square Error(MSE) loss and optimizer as Adam having parameters

$\beta_1 = 0.9$ and $\beta_2 = 0.999$ and learning rate 0.001. In order to validate the network, a validation routine was created where the predicted labels and the predicted bounding box coordinates were obtained via. model evaluation. These predicted labels and the box coordinates were then used to calculate the confusion matrix and the mean IoU(Intersection over Union) respectively. Fig. 3 shows the training loss and the confusion matrix for *HW5Net* with MSE Loss for box regression. The classification accuracy was about **88.28%**. The mean IoU on the validation set was obtained as **0.436**.

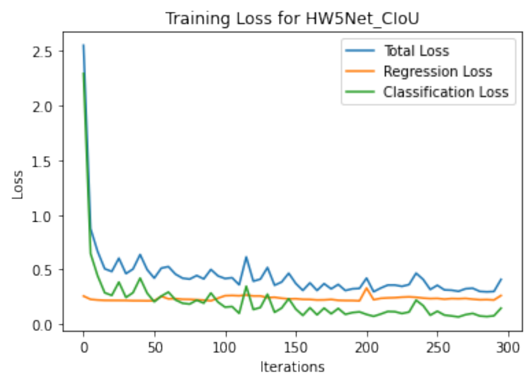


(a) Training loss vs Iterations for *HW5Net* with MSE Loss

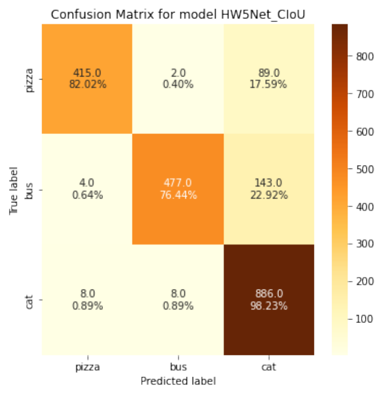


(b) Confusion Matrix for *HW5Net* with MSE Loss validations. Accuracy = 88.28%

Figure 3: Training loss and the confusion matrix for *HW5Net* with MSE Loss



(a) Training loss vs Iterations for *HW5Net* with CIoU Loss



(b) Confusion Matrix for *HW5Net* with CIoU Loss validations. Accuracy = 87.5%

Figure 4: Training loss and the confusion matrix for *HW5Net* with CIoU Loss

- **CIoU Loss** - Using the same training routine earlier, the network was trained for 20 epochs with regression loss as the Complete IoU(CIoU) loss and optimizer as Adam having parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and learning rate 0.001. In order to validate the network, the validation routine was employed where the predicted labels and the predicted bounding box coordinates were obtained via. model evaluation. These predicted labels and the box coordinates were then

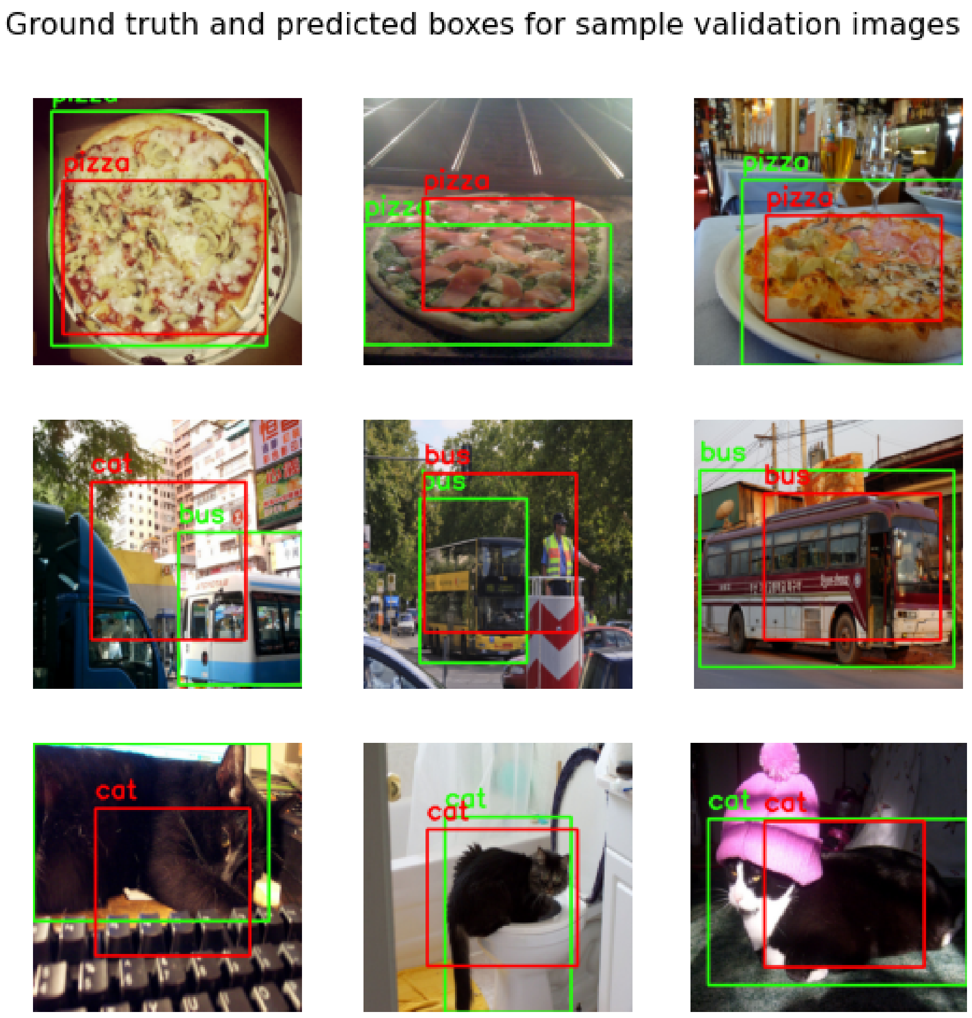


Figure 5: Sample Predicted Bounding Boxes using *HW5Net* with MSE Loss.(Green = Ground Truth Box, Red = Predicted Box). Mean IoU = 0.436.

used to calculate the confusion matrix and the mean IoU(Intersection over Union) respectively. Fig. 4 shows the training loss and the confusion matrix for *HW5Net* with CIoU Loss for box regression. The classification accuracy was about **87.5%**. The mean IoU on the validation set was obtained as **0.462**.

- **Results & Comparison**: The validation accuracies of the HW5Net network using different loss functions(MSE and CIoU) for bounding box regression on a subset of the COCO image dataset were 88.28% and 87.5% respectively. It can be seen that the classification accuracies are quite similar in the two cases as in both cases *CrossEntropy* loss was used to calculate the classification loss. It can be noticed that with CIoU loss, the mean IoU is better than with MSE loss. This is because the CIoU loss penalizes more if the predicted box does not overlap with the ground truth box. It can be seen that the network performs fairly well in classification. However, it does not do well in regression. Some of the issues in regression which was encountered were that regression values predicted went beyond the range [0,1]. This can be handled by clipping the values and

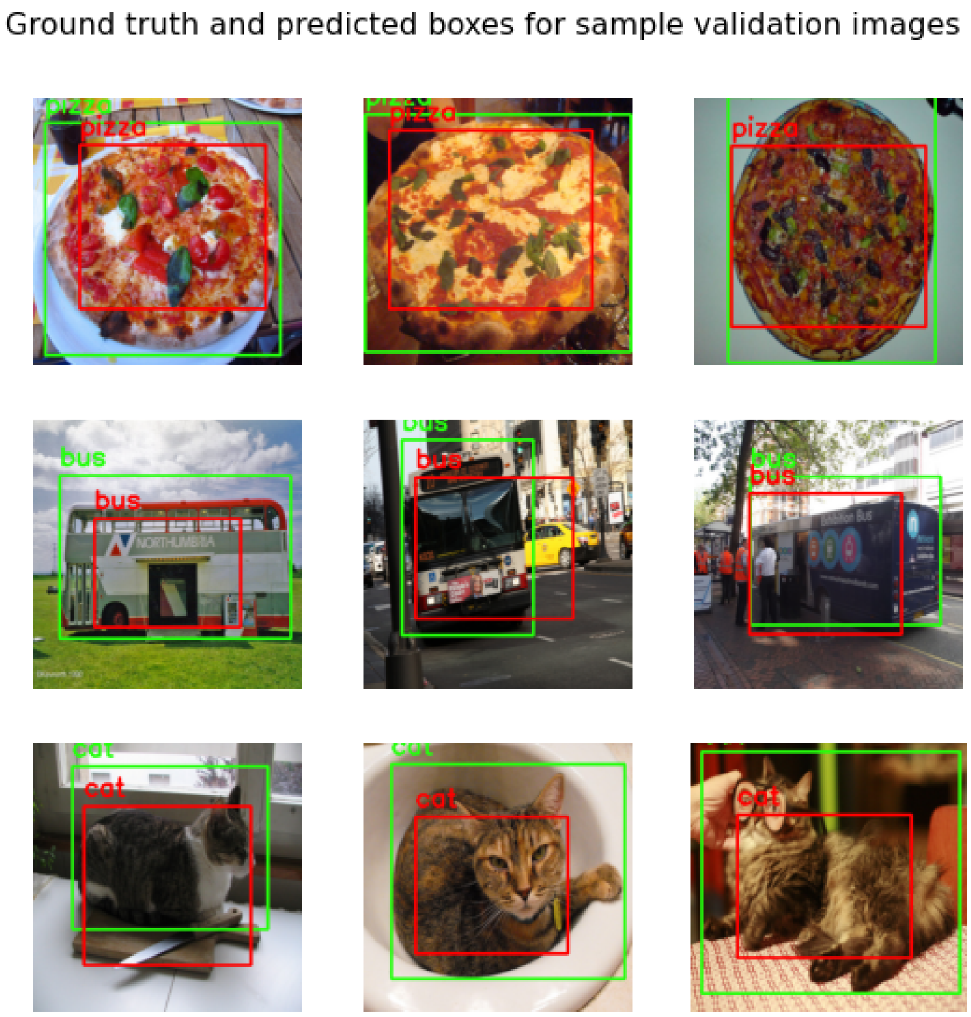

Ground truth and predicted boxes for sample validation images
pizza
pizza
pizza
pizza
pizza
bus
bus
bus
bus
bus
bus
cat
cat
cat
cat
cat

Figure 6: Sample Predicted Bounding Boxes using *HW5Net* with CIoU Loss. (Green = Ground Truth Box, Red = Predicted Box). Mean IoU = 0.462.

restricting the predictions within the desired range. Also, while using CIoU loss in some cases, the predicted coordinates of the bottom right corner were found to be lesser than that of the top left corner which caused issues in calculating the mean IoU of the predicted bounding boxes. Moreover, due to resource and time constraints, the models were run only up to 20 epochs hence it was not able to provide the desired level of accuracy. Different learning rates could also be used to train the box head and class head modules which might lead to faster training and better performance.

4 Conclusion

In conclusion, skip connections in Convolutional Neural Networks is a powerful way to capture image features along with convolutional layers which makes the network perform well in object detection and localization. It enables the deeper layers to learn identity mappings. However, the SkipBlocks should be designed carefully so that information is not lost due to downsampling. Appropriate normalization of the data is also required for training otherwise the model may face vanishing or exploding gradient problems. Hence, well-designed network architectures containing all these layer elements along with appropriate hyper-parameters are essential in making the training much more efficient and stable and getting higher classification and regression accuracy.

5 Source Code

```
# -*- coding: utf-8 -*-
"""hw5_SouradipPal.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1I3_Em9qnTJ3Pm2k_C0P1h0-b9a1oJw_S
"""

from google.colab import drive
drive.mount('/content/drive')

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/drive/MyDrive/Purdue/ECE60146/

!wget -O DLStudio-2.2.2.tar.gz \
    https://engineering.purdue.edu/kak/distDLS/DLStudio-2.2.2.tar.gz?download

!tar -xvf DLStudio-2.2.2.tar.gz

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/drive/MyDrive/Purdue/ECE60146/DLStudio-2.2.2

!wget -O /content/datasets_for_DLStudio.tar.gz \
    https://engineering.purdue.edu/kak/distDLS/datasets_for_DLStudio.tar.gz

!tar -xvf /content/datasets_for_DLStudio.tar.gz -C /content/drive/MyDrive/Purdue/ECE60146/DLStudio-2.2.2/Examples

!pip install pymsgbox

!python setup.py install

# %load_ext autoreload
# %autoreload 2

!pip install pycocotools

import os
import torch
import random
import numpy as np
import requests
import matplotlib.pyplot as plt

from tqdm import tqdm
from PIL import Image
from pycocotools.coco import COCO

seed = 0
random.seed(seed)
np.random.seed(seed)

from DLStudio import *

# Commented out IPython magic to ensure Python compatibility.
# %cd /content/drive/MyDrive/Purdue/ECE60146/DLStudio-2.2.2/Examples

!python playing_with_skip_connections.py

!python object_detection_and_localization.py
```

```
!mkdir /content/drive/MyDrive/Purdue/ECE60146/coco
!wget --no-check-certificate http://images.cocodataset.org/annotations/annotations_trainval2014.zip \
    -O /content/drive/MyDrive/Purdue/ECE60146/coco/annotations_trainval2014.zip

!unzip /content/drive/MyDrive/Purdue/ECE60146/coco/annotations_trainval2014.zip -d /content/drive/MyDrive/Purdue/ECE60146/HW5/data/

import skimage
import skimage.io as io
import cv2

input_json = '/content/drive/MyDrive/Purdue/ECE60146/HW5/data/annotations/instances_train2014.json'
class_list = ['pizza']
# ###########################
# Mapping from COCO label to Class indices
coco_labels_inverse = {}
coco = COCO(input_json)
catIds = coco.getCatIds(catNms = class_list)
categories = coco.loadCats(catIds)

categories.sort(key = lambda x: x['id'])
print(categories)

for idx, in_class in enumerate(class_list):
    for c in categories:
        if c['name'] == in_class:
            coco_labels_inverse[c['id']] = idx
print(coco_labels_inverse)

# #############################
# Retrieve Image list
imgIds = coco.getImgIds(catIds = catIds)

# #############################
# Display one random image with annotation
idx = np.random.randint(0, len(imgIds))
img = coco.loadImgs(imgIds[idx])[0]
I = io.imread(img['coco_url'])
if len(I.shape) == 2:
    I = skimage.color.gray2rgb(I)
annIds = coco.getAnnIds(imgIds = img['id'], catIds = catIds, iscrowd = False)
anns = coco.loadAnns(annIds)
fig, ax = plt.subplots(1, 1)
image = np.uint8(I)
for ann in anns :
    [x, y, w, h] = ann['bbox']
    label = coco_labels_inverse[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, class_list[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()

# !mkdir /content/drive/MyDrive/Purdue/ECE60146/HW5/data/

# !rm -rf /content/drive/MyDrive/Purdue/ECE60146/HW5/data/train2014
# !mkdir /content/drive/MyDrive/Purdue/ECE60146/HW5/data/train2014

# !rm -rf /content/drive/MyDrive/Purdue/ECE60146/HW5/data/val2014
# !mkdir /content/drive/MyDrive/Purdue/ECE60146/HW5/data/val2014
# !mkdir /content/drive/MyDrive/Purdue/ECE60146/HW5/saved_models

# Image Downloader class to download COCO images
class ImageDownloader():
    def __init__(self, root_dir, annotation_path, classes):
        self.root_dir = root_dir
        self.annotation_path = annotation_path
        self.classes = classes
        self.class_dir = {}
        self.coco = COCO(self.annotation_path)

    # Create directories same as category names to save images
    def create_dir(self):
        for c in self.classes:
            dir = os.path.join(self.root_dir, c)
            self.class_dir[c] = dir
            if not os.path.exists(dir):
                os.makedirs(dir)

    # Download images
    def download_images(self, download = True, val = False):
        img_paths = {}
        img_bboxes = {}
        for c in tqdm(self.classes):
            class_id = self.coco.getCatIds(c)
            img_id = self.coco.getImgIds(catIds=class_id)
            imgs = self.coco.loadImgs(img_id)

            img_paths[c] = []
            img_bboxes[c] = []

            for i, id in enumerate(img_id):
                annIds = self.coco.getAnnIds(imgIds = id, catIds = class_id, iscrowd = False)
                anns = self.coco.loadAnns(annIds)
                if len(anns) == 1:
                    bbox = anns[0]['bbox']
                    if bbox[2]*bbox[3] >= 40000:
                        img_path = os.path.join(self.root_dir, c, imgs[i]['file_name'])
                        if download:
                            done = self.download_image(img_path, imgs[i]['coco_url'])
                            if done:
                                self.convert_image(img_path)
                                img_paths[c].append(img_path)
                                img_bboxes[c].append(bbox)
                        else:
                            img_paths[c].append(img_path)
                            img_bboxes[c].append(bbox)

        return img_paths, img_bboxes

    # Download image from URL using requests
    def download_image(self, path, url):
        try:
            img_data = requests.get(url).content
            with open(path, 'wb') as f:
                f.write(img_data)
            return True
        except Exception as e:
            print(f"Caught exception: {e}")
        return False

    # Convert image
    def convert_image(self, path):
        im = Image.open(path)
        if im.mode != "RGB":
            im = im.convert(mode="RGB")

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

# Download training images
train_downloader = ImageDownloader('/content/drive/MyDrive/Purdue/ECE60146/HW5/data/train2014',
                '/content/drive/MyDrive/Purdue/ECE60146/HW5/data/annotations/instances_train2014.json',
                classes)
train_downloader.create_dir()
train_img_paths, train_img_bboxes = train_downloader.download_images(download = False)

len(train_img_paths['pizza'])

# Download validation images
val_downloader = ImageDownloader('/content/drive/MyDrive/Purdue/ECE60146/HW5/data/val2014',
                '/content/drive/MyDrive/Purdue/ECE60146/HW5/data/annotations/instances_val2014.json',
                classes)
val_downloader.create_dir()
val_img_paths, val_img_bboxes = val_downloader.download_images(download = False, val = True)

len(val_img_paths['bus'])

# Plotting sample training images
fig, axes = plt.subplots(3, 3, figsize=(9, 9))

for i, cls in enumerate(classes):
    for j, path in enumerate(train_img_paths[cls][:3]):
        im = Image.open(path).convert('RGB')
        im = np.ascontiguousarray(im, dtype=np.uint8)
        [x, y, w, h] = train_img_bboxes[cls][j]
        im = cv2.rectangle(im, (int(x), int(y)), (int(x + w), int (y + h)), (36, 255, 12), 2)
        im = cv2.putText(im, cls, (int(x), int(y - 10)), cv2.FONT_HERSHEY_SIMPLEX,0.8, (36, 255, 12), 2)
        axes[i][j].imshow(im)
        axes[i][j].set_title(cls)

fig.suptitle('Sample training images from COCO dataset', fontsize=16, y=0.92)
plt.show()

# Plotting sample validation images
fig, axes = plt.subplots(3, 3, figsize=(9, 9))

for i, cls in enumerate(classes):
    for j, path in enumerate(val_img_paths[cls][:3]):
        im = Image.open(path).convert('RGB')
        im = np.ascontiguousarray(im, dtype=np.uint8)
        [x, y, w, h] = val_img_bboxes[cls][j]
        im = cv2.rectangle(im, (int(x), int(y)), (int(x + w), int (y + h)), (36, 255, 12), 2)
        im = cv2.putText(im, cls, (int(x), int(y - 10)), cv2.FONT_HERSHEY_SIMPLEX,0.8, (36, 255, 12), 2)
        axes[i][j].imshow(im)
        axes[i][j].set_title(cls)

fig.suptitle('Sample validation images from COCO dataset', fontsize=16, y=0.95)
plt.show()

import os
import torch

# Custom dataset class for COCO
class CocoObjectDetectionDataset(torch.utils.data.Dataset):
    def __init__(self, root, paths, bboxes, transforms=None):
        super().__init__()
        self.root_dir = root
        self.paths = paths
        self.bboxes = bboxes
        self.classes = os.listdir(self.root_dir)
        self.transforms = transforms
        self.img_paths = []
        self.img_labels = []
        self.img_bboxes = []
        self.class_to_idx = {'pizza':0, 'bus':1, 'cat':2}
        self.idx_to_class = {i:c for c, i in self.class_to_idx.items()}

        for cls in self.classes:
            self.img_paths+=paths[cls]
            self.img_labels+=[self.class_to_idx[cls]]*len(paths[cls])
            self.img_bboxes+=bboxes[cls]

    def __len__(self):
        # Return the total number of images
        return len(self.img_paths)

    def __getitem__(self, index):
        index = index % len(self.img_paths)
        img_path = self.img_paths[index]
        img_label = self.img_labels[index]

        img = Image.open(img_path).convert('RGB')
        img_bbox = self.get_bbox(img, self.img_bboxes[index])
        bbox_tensor = torch.tensor(img_bbox, dtype=torch.float)

        img_transformed = self.transforms(img)
        return img_transformed, img_label, bbox_tensor

    def get_bbox(self, im, bbox):
        W, H = im.size
        [x, y, w, h] = bbox
        [x1, y1, x2, y2] = [x/W, y/W, (x+w)/W, (y+h)/H]
        return [x1, y1, x2, y2]

import torchvision.transforms as tvt

reshape_size = 256
transforms = tvt.Compose([
    tvt.Resize((reshape_size, reshape_size)),
    tvt.ToTensor(),
    tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

# Create custom training dataset
train_dataset = CocoObjectDetectionDataset('/content/drive/MyDrive/Purdue/ECE60146/HW5/data/train2014/',
                                           train_img_paths, train_img_bboxes, transforms=transforms)

len(train_dataset)

# Create custom validation dataset
val_dataset = CocoObjectDetectionDataset('/content/drive/MyDrive/Purdue/ECE60146/HW5/data/val2014/',
                                         val_img_paths, val_img_bboxes, transforms=transforms)

len(val_dataset)

train_dataset[0][2]

# Create custom training/validation dataloader
train_data_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=2)
val_data_loader = torch.utils.data.DataLoader(val_dataset, batch_size=64, shuffle=False, num_workers=2)

# Check dataloader
train_loader_iter = iter(train_data_loader)
img, label, bbox  = next(train_loader_iter)

print('img has length: ', len(img))
print('target has length: ', len(label))
print(img[0].shape)

# Commented out IPython magic to ensure Python compatibility.
# Routine to train a neural network
def train_net(device, net, optimizer, criterion_cls, criterion_reg, data_loader,
              model_name, epochs = 10, display_interval = 100):
    net = net.to(device)

    loss_running_record = []
    loss_reg_running_record = []
    loss_cls_running_record = []

    for epoch in range(epochs):
        running_loss = 0.0
        running_loss_reg = 0.0
        running_loss_cls = 0.0
        for i, data in enumerate(data_loader):
            inputs, labels, bboxes = data
            inputs = inputs.to(device)
            labels = labels.to(device)
            bboxes = bboxes.to(device)
            optimizer.zero_grad()
            outputs = net(inputs)
            output_label = outputs[0]
            output_bbox = outputs[1]

            # adding the classification and regression losses
            loss_cls = criterion_cls(output_label, labels)
            loss_reg = criterion_reg(output_bbox, bboxes)
            loss = loss_reg + loss_cls

            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            running_loss_reg += loss_reg.item()
            running_loss_cls += loss_cls.item()

            if (i+1) % display_interval == 0:
                avg_loss = running_loss / display_interval
                avg_loss_reg = running_loss_reg / display_interval
                avg_loss_cls = running_loss_cls / display_interval
                print ("[epoch : %d, batch : %5d] loss : %.3f,\t\t loss_reg: %.3f, \t\t loss_cls: %.3f" \
#                         % (epoch + 1, i + 1, avg_loss, avg_loss_reg, avg_loss_cls))
                loss_running_record.append(avg_loss)
                loss_reg_running_record.append(avg_loss_reg)
                loss_cls_running_record.append(avg_loss_cls)
            running_loss = 0.0
            running_loss_reg = 0.0
            running_loss_cls = 0.0

    checkpoint_path = os.path.join('/content/drive/MyDrive/Purdue/ECE60146/HW5/saved_models',
                                  f'{model_name}.pth')
    torch.save(net.state_dict(), checkpoint_path)

    return loss_running_record, loss_reg_running_record, loss_cls_running_record

# Plotting training loss
def plot_loss(loss, loss_reg, loss_cls, display_interval, model_name):
    plt.figure()
    plt.plot(np.arange(len(loss))*display_interval, loss, label="Total Loss");
    plt.plot(np.arange(len(loss_reg))*display_interval, loss_reg, label="Regression Loss");
    plt.plot(np.arange(len(loss_cls))*display_interval, loss_cls, label="Classification Loss");
    plt.title(f'Training Loss for {model_name}')
    plt.xlabel('Iterations')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

# Commented out IPython magic to ensure Python compatibility.
# Routine to validate a neural network
def validate_net(device, net, criterion_cls, criterion_reg,
                 data_loader, model_path = None):
    if model_path is not None:
        net.load_state_dict(torch.load(model_path, map_location=torch.device('cpu')))
    net = net.to(device)
    net.eval()
    running_loss = 0.0
    running_loss_reg = 0.0
    running_loss_cls = 0.0

    device_cpu = torch.device('cpu')

    iters = 0
    imgs = []
    all_labels = []
    all_labels_pred = []

    all_bboxes = []
    all_bboxes_pred = []

    with torch.no_grad():
        for i, data in enumerate(data_loader):
            inputs, labels, bboxes = data
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = net(inputs)
            bboxes = bboxes.to(device)
            optimizer.zero_grad()
            outputs = net(inputs)
            output_label = outputs[0]
            output_bbox = outputs[1]

            # adding the classification and regression losses
            loss_cls = criterion_cls(output_label, labels)
            loss_reg = criterion_reg(output_bbox, bboxes)
            loss = loss_reg + loss_cls

            running_loss += loss.item()
            running_loss_reg += loss_reg.item()
            running_loss_cls += loss_cls.item()
            iters += 1
            pred_labels = torch.argmax(output_label.data, axis = 1)

            imgs += list(inputs.to(device_cpu))
            all_labels += list(labels.to(device_cpu).numpy())
            all_bboxes += list(bboxes.to(device_cpu).numpy())
            all_labels_pred += list(pred_labels.to(device_cpu).numpy())
            all_bboxes_pred += list(output_bbox.to(device_cpu).numpy())


        print ("validation loss : %.3f,\t validation loss_reg: %.3f, \t validation loss_cls: %.3f" \
#                        % (running_loss/iters, running_loss_reg/iters, running_loss_cls/iters))
```

```
    return imgs, all_labels, all_labels_pred, all_bboxes, all_bboxes_pred

# Function to calculate confusion matrix
def calc_confusion_matrix(num_classes, actual, predicted):
    conf_mat = np.zeros((num_classes, num_classes))
    for a, p in zip(actual, predicted):
        conf_mat[a][p]+=1
    return conf_mat

# Function to calculate mean IoU
from torchvision.ops import complete_box_iou

def calc_mean_IoU(gt_bboxes, pred_bboxes):
    iou = 0
    gt_bboxes, pred_bboxes = np.array(gt_bboxes), np.array(pred_bboxes)
    for gt_bbox, pred_bbox in zip(gt_bboxes, pred_bboxes):
        pred_bboxes[pred_bboxes>1] = 1
        pred_bboxes[pred_bboxes<0] = 0
        iou += complete_box_iou(torch.Tensor(gt_bbox).unsqueeze(0), torch.Tensor(pred_bbox).unsqueeze(0))[0][0]
    iou/= gt_bboxes.shape[0]
    return iou

import seaborn as sns

# Plotting confusion matrix
def plot_conf_mat(conf_mat, classes, model_name):
    labels = []
    num_classes = len(classes)
    for row in range(num_classes):
        rows = []
        total_labels =  np.sum(conf_mat[row])
        for col in range(num_classes):
            count = conf_mat[row][col]
            percent = "%.2f%%" % (count / total_labels * 100)
            label = str(count) + '\n' + str(percent)
            rows.append(label)
        labels.append(rows)
    labels = np.asarray(labels)

    plt.figure(figsize=(6, 6))
    sns.heatmap(conf_mat, annot=labels, fmt="", cmap="YlOrBr", cbar=True,
                xticklabels=classes, yticklabels=classes)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title(f'Confusion Matrix for model {model_name}')
    plt.show()

# Defining ResBlock
import torch.nn as nn
import torch.nn.functional as F

class ResBlock(nn.Module):
    def __init__(self, in_ch, out_ch, downsample=False):
        super(ResBlock, self).__init__()
        self.downsample = downsample
        self.in_ch = in_ch
        self.out_ch = out_ch
        self.conv1 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(out_ch)
        self.bn2 = nn.BatchNorm2d(out_ch)
        self.dropout = nn.Dropout(0.2)
        self.relu = nn.LeakyReLU()
        if downsample:
            self.downsampler = nn.Conv2d(in_ch, out_ch, 1, stride=2)

    def forward(self, x):
        identity = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        if self.in_ch == self.out_ch:
            out = self.conv2(out)
            out = self.bn2(out)
            out = self.relu(out)
            out = self.dropout(out)

        if self.downsample:
            out = self.downsampler(out)
            identity = self.downsampler(identity)

        if self.in_ch == self.out_ch:
            out = out + identity
        else:
            out[:,:self.in_ch,:,:] += identity
            out[:,self.in_ch:,:,:] += identity
        return out

# Define HW5Net architecture
class HW5Net(nn.Module):
    """ Resnet - based encoder that consists of a few
    downsampling + several Resnet blocks as the backbone
    and two prediction heads .
    """

    def __init__ (self, input_nc, ngf = 8, n_blocks = 4):
        """
        Parameters :
        input_nc (int) -- the number of channels input images
        output_nc (int) -- the number of channels output images
        ngf (int ) -- the number of filters in the first conv layer
        n_blocks (int) -- the number of ResNet blocks
        """
        assert(n_blocks >= 0)
        super(HW5Net, self). __init__ ()
        # The first conv layer
        model = [
            nn.ReflectionPad2d(3),
            nn.Conv2d(input_nc, ngf, kernel_size = 7, padding = 0),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True)
        ]
        # Add downsampling layers
        n_downsampling = 4
        for i in range(n_downsampling):
            mult = 2 ** i
            model += [
                nn.Conv2d(ngf * mult, ngf * mult * 2, kernel_size = 3, stride = 2, padding = 1),
                nn.BatchNorm2d(ngf * mult * 2),
                nn.ReLU(True)
            ]
        # Add your own ResNet blocks
        mult = 2 ** n_downsampling
        for i in range(n_blocks):
            model += [ResBlock(ngf * mult, ngf * mult, downsample = False)]
        self.model = nn.Sequential(*model)
        # The classification head
        class_head = [
            nn.Linear(128*16*16, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.2),
            nn.Linear(512, 64),
            nn.ReLU(inplace=True),
            nn.Linear(64, 3)
        ]
        self.class_head = nn.Sequential(*class_head)
        # The bounding box regression head
        bbox_head = [
            nn.Linear(128*16*16, 2048),
            nn.ReLU(inplace=True),
            nn.Linear(2048, 512),
            nn.ReLU(inplace=True),
            nn.Linear(512, 4)
        ]
        self.bbox_head = nn.Sequential(*bbox_head)

    def forward (self, input):
        ft = self.model(input)
        ft = ft.view(-1, 128*16*16)
        cls = self.class_head(ft)
        bbox = self.bbox_head(ft)
        return cls, bbox

# Initialize device
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
device

# Initialize HW5Net with MSE Loss
net = HW5Net(3)

# net.load_state_dict(torch.load('/content/drive/MyDrive/Purdue/ECE60146/HW5/saved_models/HW5Net_MSE.pth',
#                                  map_location=torch.device('cpu')))

criterion_cls = torch.nn.CrossEntropyLoss()
criterion_reg = torch.nn.MSELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3, betas=(0.9, 0.999))
epochs = 10
display_interval = 5

# Display Number of Layers
num_layers = len(list(net.parameters()))
print(num_layers)

# Display Number of Trainable Parameters
num_params = sum(p.numel() for p in net.parameters() if p.requires_grad)
print(num_params)

# Train HW5Net with MSE Loss
net1_losses = train_net(device, net, optimizer=optimizer, criterion_cls=criterion_cls,
                        criterion_reg=criterion_reg, data_loader = train_data_loader,
                        model_name = 'HW5Net_MSE', epochs=epochs, display_interval = display_interval)

# Plotting HW5Net_MSE training loss
plot_loss(net1_losses[0], net1_losses[1], net1_losses[2], display_interval, 'HW5Net_MSE')

# Validate HW5Net with MSE Loss
save_path = '/content/drive/MyDrive/Purdue/ECE60146/HW5/saved_models/HW5Net_MSE.pth'
imgs, gt_label, pred_label, gt_bboxes, pred_bboxes = validate_net(device, net, criterion_cls,
                        criterion_reg, val_data_loader, model_path = save_path)

# Calculate HW5Net_MSE confusion matrix and accuracy
conf_mat = calc_confusion_matrix(3, gt_label, pred_label)
print(conf_mat)
accuracy = np.trace(conf_mat) / float(np.sum(conf_mat))
print(accuracy)

# Plotting HW5Net_MSE confusion matrix
plot_conf_mat(conf_mat, classes, 'HW5Net_MSE')

# Calculating mean IoU for HW5Net_MSE
calc_mean_IoU(gt_bboxes, pred_bboxes)

# Plotting ground truth and predicted boxes for validation images with HW5Net_MSE
fig, axes = plt.subplots(3, 3, figsize=(9, 9))

ind0 = [i for i, l in enumerate(gt_label) if l == 0]
ind1 = [i for i, l in enumerate(gt_label) if l == 1]
ind2 = [i for i, l in enumerate(gt_label) if l == 2]
indices = [ind0[:3], ind1[:3], ind2[:3]]

for i in range(3):
    for j in range(3):
        ind = indices[i][j]
        I = imgs[ind].numpy()
        I = (I*0.5+0.5)*255
        label = int(gt_label[ind])
        pred = int(pred_label[ind])
        image = np.ascontiguousarray(I.transpose(1,2,0), dtype=np.uint8)
        [x1, y1, x2, y2] = gt_bboxes[ind]*256.0
        [X1, Y1, X2, Y2] = pred_bboxes[ind]*256.0
        image = cv2.rectangle(image, (int(x1), int(y1)), (int(x2), int(y2)), (36, 255, 12), 2)
        image = cv2.putText(image, classes[label], (int(x1), int(y1 - 10)), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (36, 255, 12), 2)
        image = cv2.rectangle(image, (int(X1), int(Y1)), (int(X2), int(Y2)), (255, 0, 0), 2)
        image = cv2.putText(image, classes[pred], (int(X1), int(Y1 - 10)), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 0, 0), 2)
        axes[i][j].imshow(image)
        axes[i][j].set_axis_off()
```

```
fig.suptitle('Ground truth and predicted boxes for sample validation images', fontsize=16, y=0.95)
plt.axis('tight')
plt.show()

# Define CIoU Loss
from torchvision.ops import complete_box_iou_loss
class CIoULoss(nn.Module):
    def __init__(self):
        super(CIoULoss, self).__init__()

    def forward(self, outputs, targets):
        loss = complete_box_iou_loss(outputs, targets, reduction = 'mean')
        return loss

# Initialize HW5Net with CIoU Loss
net2 = HW5Net(3)

# net2.load_state_dict(torch.load('/content/drive/MyDrive/Purdue/ECE60146/HW5/saved_models/HW5Net_CIoU.pth',
#                                  map_location=torch.device('cpu')))

net2 = net2.to(device)
criterion_cls = torch.nn.CrossEntropyLoss()
criterion_reg = CIoULoss()
optimizer = torch.optim.Adam(net2.parameters() , lr=1e-3, betas=(0.9, 0.999))
epochs = 5
display_interval = 5

# Train HW5Net with CIoU Loss
net2_losses = train_net(device, net2, optimizer=optimizer, criterion_cls=criterion_cls,
                        criterion_reg=criterion_reg, data_loader = train_data_loader,
                        model_name = 'HW5Net_CIoU', epochs=epochs, display_interval = display_interval)

# Plotting HW5Net_CIoU training loss
plot_loss(net2_losses[0], net2_losses[1], net2_losses[2], display_interval, 'HW5Net_CIoU')

save_path = '/content/drive/MyDrive/Purdue/ECE60146/HW5/saved_models/HW5Net_CIoU.pth'
imgs, gt_label, pred_label, gt_bboxes, pred_bboxes = validate_net(device, net2, criterion_cls,
                        criterion_reg, val_data_loader, model_path = save_path)

conf_mat = calc_confusion_matrix(3, gt_label, pred_label)
print(conf_mat)
accuracy = np.trace(conf_mat) / float(np.sum(conf_mat))
print(accuracy)

plot_conf_mat(conf_mat, classes, 'HW5Net_CIoU')

# Calculating mean IoU for HW5Net_CIoU
calc_mean_IoU(gt_bboxes, pred_bboxes)

# Plotting ground truth and predicted boxes for validation images for HW5Net_CIoU
fig, axes = plt.subplots(3, 3, figsize=(9, 9))

ind0 = [i for i, l in enumerate(gt_label) if l == 0]
ind1 = [i for i, l in enumerate(gt_label) if l == 1]
ind2 = [i for i, l in enumerate(gt_label) if l == 2]
indices = [ind0[100:103], ind1[100:103], ind2[100:103]]

for i in range(3):
    for j in range(3):
        ind = indices[i][j]
        I = imgs[ind].numpy()
        I = (I*0.5+0.5)*255
        print(I.shape)
        label = int(gt_label[ind])
        pred = int(pred_label[ind])
        image = np.ascontiguousarray(I.transpose(1,2,0), dtype=np.uint8)
        [x1, y1, x2, y2] = gt_bboxes[ind]*256.0
        [X1, Y1, X2, Y2] = pred_bboxes[ind]*256.0
        image = cv2.rectangle(image, (int(x1), int(y1)), (int(x2), int(y2)), (36, 255, 12), 2)
        image = cv2.putText(image, classes[label], (int(x1), int(y1 - 10)), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (36, 255, 12), 2)
        image = cv2.rectangle(image, (int(X1), int(Y1)), (int(X2), int(Y2)), (255, 0, 0), 2)
        image = cv2.putText(image, classes[pred], (int(X1), int(Y1 - 10)), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 0, 0), 2)
        axes[i][j].imshow(image)
        axes[i][j].set_axis_off()

fig.suptitle('Ground truth and predicted boxes for sample validation images', fontsize=16, y=0.95)
```

```
plt.axis('tight')
plt.show()
```