ECE 60146 Deep Learning
Homework 5
03/06/2023

Name: *Nikita Ravi (ravi30)*

1 Introduction

The objective of this homework assignment was to understand the purpose of skip blocks and its usage in object detection and localization. With skip connections, a network includes shortcut pathways so that the loss calculated at the output can be felt more strongly in the earlier layers of the network during back-propagation. Layers of convolutional networks and a final linear layer is used to obtain the classification label and predicted bounding box. Two loss functions are necessary to measure the regression loss (i.e. MSE and Complete Box IoU Loss) for the bounding box and cross-entropy loss for the classification label. IoU stands for Intersection over Union and it takes the ratio of the intersection of the two bounding boxes to their union.

2 Methodology

2.1 COCO Downloader

The script below was used to load all the images and save them into its respective categories and store its class label and the provided bounding box into a dataframe. The bounding box saved into the dataframe is that of the one surrounding the dominant object and has an area greater than 200×200 pixels.

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

# Global Variables
train_directory = r"/Users/nikitaravi/Documents/Academics/ECE 60146/HW5/
    train2014"
val_directory = r"/Users/nikitaravi/Documents/Academics/ECE 60146/HW5/
    val2014"
train_annotations = r"/Users/nikitaravi/Documents/Academics/ECE 60146/HW5/
    annotations/instances_train2014.json"
val_annotations = r"/Users/nikitaravi/Documents/Academics/ECE 60146/HW5/
    annotations/instances_val2014.json"
coco_train = COCO(train_annotations)
coco_val = COCO(val_annotations)
categories = ["pizza", "bus", "cat"]
columns = ["id", "category", "path_to_image", "x1", "y1", "x2", "y2"]
```

```
new_image_size = 256
resize_image = tvt.Compose([tvt.Resize((new_image_size,new_image_size))])


def parser():
    parser = argparse.ArgumentParser(description="Load data into dataframe")
    parser.add_argument("--train", action="store_true", help="Download training data or testing data")
    args = parser.parse_args()
    return args

def create_dataframe(train=True):
    saved_count = 0
    df = pd.DataFrame(columns=columns)
    ids, class_list, paths_to_image, bbox_x1s, bbox_y1s, bbox_x2s, bbox_y2s = [], [], [], [], [], [], []
    directory = train_directory if train else val_directory
    new_dirname = f"./train" if train else f"./val"
    coco = coco_train if train else coco_val
    os.makedirs(new_dirname, exist_ok=True)
    for class_idx, category in enumerate(categories):
        category_id = coco.getCatIds(catNms=category)
        image_ids = coco.getImgIds(catIds=category_id)
        for img_id in image_ids:
            current_image = coco.loadImgs(ids=img_id)
            current_id = current_image[0]["id"]
            annotation_id = coco.getAnnIds(imgIds=img_id, catIds=category_id, iscrowd=False)
            annotation = coco.loadAnns(ids=annotation_id)

            check_bbox = False
            if(annotation[0]["area"] > 200 * 200):
                saved_count += 1
                check_bbox = True
            if(check_bbox):
                path_to_image = current_image[0]["file_name"]
                original_width, original_height = current_image[0]["width"], current_image[0]["height"]

                # Resize Image
                original_image = resize_image(read_image(os.path.join(directory, path_to_image)))
                original_image = original_image.repeat(3,1,1) if original_image.size()[0] == 1 else original_image # Check if image has three channels
                original_image = tvt.functional.to_pil_image(original_image).convert("RGB") # convert to RGB image

                # Resize bbox
                annotation_width, annotation_height = annotation[0]["bbox"][2], annotation[0]["bbox"][3]
                annotation_left_x, annotation_left_y = annotation[0]["bbox"][0], annotation[0]["bbox"][1]
```

```
                x_scale = new_image_size / original_width
                y_scale = new_image_size / original_height

                new_bbox_width = x_scale * annotation_width
                new_bbox_height = y_scale * annotation_height
                new_x1 = x_scale * annotation_left_x
                new_y1 = y_scale * annotation_left_y

                new_x2 = new_x1 + new_bbox_width
                new_y2 = new_y1 + new_bbox_height

                image_filename = f"{category}_{current_id}.png"
                original_image.save(os.path.join(new_dirname,
    image_filename))

                # Add everything to the list to add to the df
                ids.append(current_id)
                class_list.append(category)
                paths_to_image.append(os.path.join(new_dirname,
    image_filename))
                bbox_x1s.append(new_x1)
                bbox_y1s.append(new_x2)
                bbox_x2s.append(new_y1)
                bbox_y2s.append(new_y2)

    df["id"] = ids
    df["category"] = class_list
    df["path_to_image"] = paths_to_image
    df["x1"] = bbox_x1s
    df["x2"] = bbox_y1s
    df["y1"] = bbox_x2s
    df["y2"] = bbox_y2s

    filename = "train_data.csv" if train else "test_data.csv"
    df.to_csv(filename)


if __name__ == "__main__":
    args = parser()
    train = True if args.train else False

    create_dataframe(train)
```

Listing 1: COCO Downloader

2.1.1 Inputs

The following image is a collection of the input images with its corresponding bounding box provided by the COCO dataset

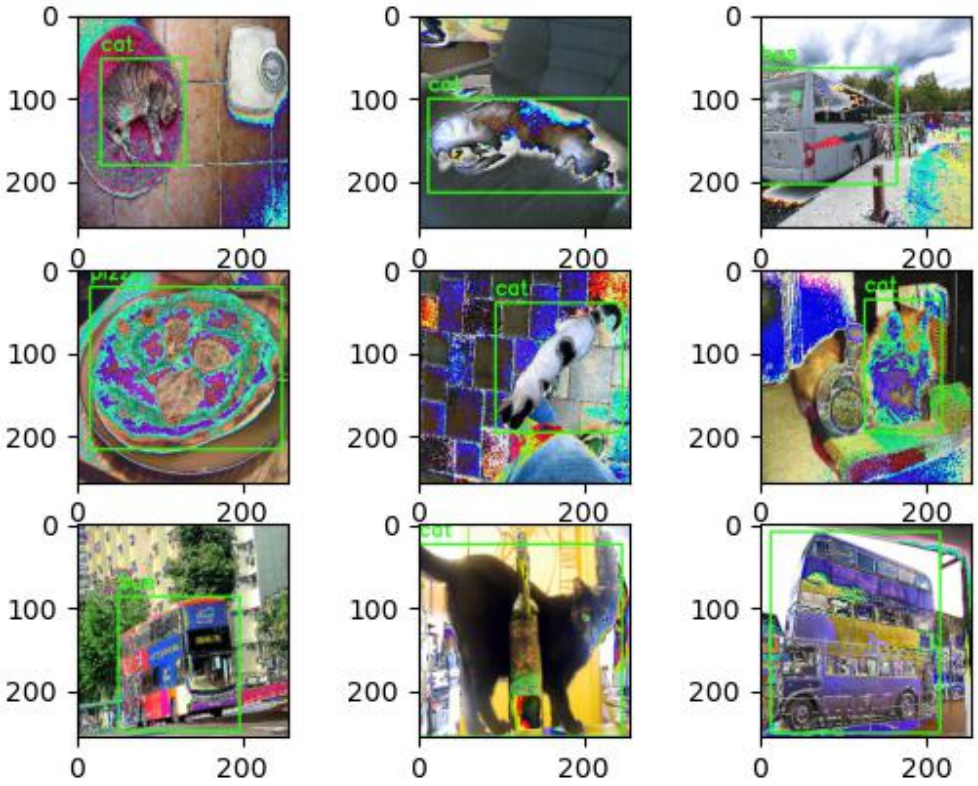

Figure 1: Inputs

2.2 Training

The methodology involved to train and test the model against the training and validation dataset respectively is as follows:

- The loss function used for classifying the images is the cross-entropy loss
- The loss function used for predicting the bounding box is the mean squared error loss or the complete box IoU loss
- The optimizer used is the Adam optimizer

- The loss graph from the cross-entropy loss function is retained to compute the bounding box regression loss with respect to that label
- Every 50 iterations, the model is saved and the average running loss is printed out

2.2.1 Source Code

```
# Import Libraries
import numpy as np
import torch
import torchvision.transforms as tvt
import torch.utils.data
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
from PIL import Image
import os
import seaborn as sns
from torchvision.ops import complete_box_iou_loss
import pandas as pd
import cv2
import argparse

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# GLOBAL VARIABLES
device = 'cuda' if torch.cuda.is_available() else 'cpu'
device = torch.device(device)
path_to_model = r"/content/drive/MyDrive/ECE 60146/HW5/model/"

################################## CREATING DATASETS
    ##############################
class GenerateDataset(torch.utils.data.Dataset):
    def __init__(self, df, transform=None):
        super().__init__()
        self.df = df
        self.transform = transform

    def __return_integer_encoding(self, category):
        categories = {"pizza": 0, "bus": 1, "cat": 2}
        return categories[category]

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

    def __getitem__(self, idx):
        image_info = self.df.iloc[idx]
        path_to_image = os.path.join(r"/content/drive/MyDrive/ECE 60146/
    HW5/", image_info["path_to_image"])
        image = Image.open(path_to_image)
        image = self.transform(image) if self.transform else image
```

```
        label = int(self.__return_integer_encoding(image_info["category"]))

        bbox = [image_info["x1"], image_info["y1"], image_info["x2"], image_info["y2"]]
        # bbox_with_pixel_coords = [bbox[0], bbox[1], bbox[0] + bbox[2], bbox[1] + bbox[3]]
        bbox_tensor = torch.tensor(bbox, dtype=torch.float)

        return image, label, bbox_tensor

def get_training_dataloader():
    # Constants
    transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

    # Create PyTorch Datasets and Dataloader
    df = pd.read_csv(r"/content/drive/MyDrive/ECE 60146/HW5/train_data.csv")
    train_dataset = GenerateDataset(df, transform)
    #print(train_dataset[0][0].shape, train_dataset[0][1], train_dataset[0][2]); quit()
    trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=2, num_workers=2, shuffle=True)
    return trainloader

def get_testing_dataloader():
    # Constants
    transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

    # Create PyTorch Datasets and Dataloader
    df = pd.read_csv(r"/content/drive/MyDrive/ECE 60146/HW5/test_data.csv")
    test_dataset = GenerateDataset(df, transform)
    testloader = torch.utils.data.DataLoader(test_dataset, batch_size=2, num_workers=2, shuffle=True)
    return testloader

############################## NETWORK ##############################
class ResnetBlock(nn.Module):
    # Inspired by Professor Kak's SkipBlock class
    def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
        super(ResnetBlock, self).__init__()
        self.in_ch = in_ch
        self.out_ch = out_ch
        self.downsample = downsample
        self.skip_connections = skip_connections

        self.conv1 = nn.Conv2d(in_ch, out_ch, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_ch, out_ch, kernel_size=3, stride=1, padding=1)
```

```
85
86          self.bn1 = nn.BatchNorm2d(out_ch)
87          self.bn2 = nn.BatchNorm2d(out_ch)
88
89          if(self.downsample):
90              self.downsampler = nn.Conv2d(in_ch, out_ch, kernel_size=1,
    stride=2)
91
92      def forward(self, x):
93          identity = x
94          out = self.conv1(x)
95          out = self.bn1(out)
96          out = torch.nn.functional.relu(out)
97
98          if(self.in_ch == self.out_ch):
99              out = self.conv2(out)
100             out = self.bn2(out)
101             out = torch.nn.functional.relu(out)
102         if(self.downsample):
103             out = self.downsampler(out)
104             identity = self.downsampler(identity)
105         if(self.skip_connections):
106             if(self.in_ch == self.out_ch):
107                 out = out + identity
108             else:
109                 out = torch.cat((out[:,:self.in_ch,:,:] + identity, out[:,
     self.in_ch:, :, :] + identity), dim=1)
110         return out
111
112 class HW5Net(nn.Module):
113     # Inspired by Professor Kak's Loadnet2
114     def __init__(self, skip_connections=True, depth=16):
115         super(HW5Net, self).__init__()
116         self.skip_connections = skip_connections
117         self.depth = depth // 2
118         self.conv = nn.Conv2d(in_channels=3, out_channels=64, kernel_size
    =3, padding=1)
119         self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
120
121         # Classification
122         self.bn1 = nn.BatchNorm2d(num_features=64)
123         self.bn2 = nn.BatchNorm2d(num_features=128)
124         self.skip64_arr = nn.ModuleList()
125         for idx in range(self.depth):
126             self.skip64_arr.append(ResnetBlock(in_ch=64, out_ch=64,
    skip_connections=self.skip_connections))
127         self.skip64ds = ResnetBlock(in_ch=64, out_ch=64, downsample=True,
    skip_connections=self.skip_connections)
128         self.skip64to128 = ResnetBlock(in_ch=64, out_ch=128,
    skip_connections=self.skip_connections)
129         self.skip128_arr = nn.ModuleList()
130         for idx in range(self.depth):
131             self.skip128_arr.append(ResnetBlock(in_ch=128, out_ch=128,
    skip_connections=self.skip_connections))
```

```
        self.skip128ds = ResnetBlock(in_ch=128, out_ch=128, downsample=True, skip_connections=self.skip_connections)

        self.fc1 = nn.Linear(in_features=32*32*128, out_features=3)
        # self.fc2 = nn.Linear(in_features=1000, out_features=3) # Outputs probability of three classes

        # Regression
        self.conv_seqn = nn.Sequential(nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
                                       nn.BatchNorm2d(num_features=64),
                                       nn.ReLU(inplace=True),
                                       nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
                                       nn.ReLU(inplace=True))

        self.fc_seqn = nn.Sequential(nn.Linear(in_features=128*128*64, out_features=4)) # Outputs [x, y, x+w, y+h]

    def forward(self, x):
        x = self.pool(nn.functional.relu(self.conv(x)))

        # Classification
        cls = x.clone()
        for idx, skip64 in enumerate(self.skip64_arr[:self.depth//4]):
            cls = skip64(cls)
        cls = self.skip64ds(cls)
        for idx, skip64 in enumerate(self.skip64_arr[self.depth//4:]):
            cls = skip64(cls)
        cls = self.bn1(cls)
        cls = self.skip64to128(cls)
        for idx, skip128 in enumerate(self.skip128_arr[:self.depth//4]):
            cls = skip128(cls)
        cls = self.bn2(cls)
        cls = self.skip128ds(cls)
        for idx, skip128 in enumerate(self.skip128_arr[self.depth//4:]):
            cls = skip128(cls)
        # print(cls.shape); quit()
        cls = cls.view(-1, 32 * 32 * 128)
        cls = self.fc1(cls)
        #cls = self.fc2(cls)

        # Regression for Bbox
        bbox = self.conv_seqn(x)
        # print(bbox.shape); quit()
        bbox = bbox.view(x.size(0), -1)
        bbox = self.fc_seqn(bbox)

        return cls, bbox

############################## TRAINING ##############################
def train(model, trainloader, path_to_model=path_to_model, mse=True, lr=1e-4, betas=(0.9, 0.99), epochs=7):
    # Train function inspired by Professor Kak's
```

```
    run_code_for_training_with_CrossEntropy_and_MSE_Losses function
    print("Training Started")
    model = model.to(device)
    num_layers = len(list(model.parameters()))
    assert num_layers >= 50, f"number of layers greater that or equal to
    50 expected, got: {num_layers}"
    print(f"Number of layers: {num_layers}")

    model_name = "model_mse.pth" if mse else "model_ciou.pth"
    path_to_model = os.path.join(path_to_model, model_name)

    cls_criterion = nn.CrossEntropyLoss()
    reg_criterion = nn.MSELoss() if mse else complete_box_iou_loss
    optimizer = torch.optim.Adam(model.parameters(), lr=lr, betas=betas)
    labeling_loss_tally = []
    regression_loss_tally = []

    for epoch in range(1, epochs+1):
        print(f"Epoch: {epoch}")
        running_loss_labeling = 0.0
        running_loss_regression = 0.0
        for batch_idx, (inputs, labels, bbox) in enumerate(trainloader):
            print(f"Batch idx: {batch_idx}")
            inputs = inputs.to(device)
            labels = labels.to(device)
            bbox = bbox.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            output_label, output_bbox = outputs[0], outputs[1]

            label_loss = cls_criterion(output_label, labels)
            label_loss.backward(retain_graph=True) # Preserve this loss
    while calculating bbox loss
            if(mse):
                bbox_loss = reg_criterion(output_bbox, bbox)
            else:
                bbox_loss = reg_criterion(output_bbox, bbox, reduction="
    mean")
            bbox_loss.backward()
            optimizer.step()

            running_loss_labeling += label_loss.item()
            running_loss_regression += bbox_loss.item()

            if(batch_idx % 50 == 49):
                avg_loss_labeling = running_loss_labeling / float(50)
                avg_loss_regression = running_loss_regression / float(50)
                labeling_loss_tally.append(avg_loss_labeling)
                regression_loss_tally.append(avg_loss_regression)
                print(f"Epoch: {epoch}/{epochs}, Iteration: {batch_idx +
    1}: Labeling Loss: {avg_loss_labeling}, Regression Loss:{
    avg_loss_regression}")
                print(f"-----Saving model-----")
```

```
                torch.save(model.state_dict(), path_to_model)

                running_loss_labeling = 0.0
                running_loss_regression = 0.0

    return labeling_loss_tally, regression_loss_tally

def test(model, testloader, path_to_model=path_to_model, mse=True, num_classes=3):
    model_name = "model_mse.pth" if mse else "model_ciou.pth"
    path_to_model = os.path.join(path_to_model, model_name)
    model.load_state_dict(torch.load(path_to_model))
    model = model.to(device)
    confusion_matrix = np.zeros((num_classes, num_classes))
    image_info = []

    with torch.no_grad():
        for inputs, labels, bbox in testloader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            bbox = bbox.to(device)

            outputs = model(inputs)
            output_label = outputs[0]
            output_bbox = outputs[1].tolist()

            _, predicted = torch.max(output_label, dim=1)
            for label, prediction in zip(labels, predicted):
                confusion_matrix[label][prediction] += 1

            for img, original_label, predicted_label, original_bbox, predicted_bbox in zip(inputs, labels, predicted, bbox, output_bbox):
                image_info.append({"Image": img, "Original Label": original_label, "Predicted Label": predicted_label,
                                   "Original Bbox": original_bbox, "Predicted Bbox": predicted_bbox})

    accuracy = np.trace(confusion_matrix) / np.sum(confusion_matrix)
    return confusion_matrix, accuracy, image_info

def plot_labeling_loss(labeling, epochs):
    iterations = range(len(labeling))
    figure = plt.figure(1)
    plt.plot(iterations, labeling, label="Labeling Loss")

    plt.title(f"Loss per Iteration")
    plt.xlabel(f"Iterations over {epochs} epochs")
    plt.ylabel("Loss")

    filename = "label_loss.jpg"
    plt.savefig(os.path.join("/content/drive/MyDrive/ECE 60146/HW5/Results", filename))

def plot_regression_loss(regression, epochs, mse):
```

```
    iterations = range(len(regression))
    figure = plt.figure(2)
    plt.plot(iterations, regression, label="Regression Loss")

    plt.title(f"Loss per Iteration")
    plt.xlabel(f"Iterations over {epochs} epochs")
    plt.ylabel("Loss")

    filename = "mse_loss.jpg" if mse else "ciou_loss.jpg"
    plt.savefig(os.path.join("/content/drive/MyDrive/ECE 60146/HW5/Results", filename))

def display_confusion_matrix(conf, accuracy, class_list=["pizza", "bus", "cat"], mse=True):
    figure = plt.figure(3)
    sns.heatmap(conf, xticklabels=class_list, yticklabels=class_list, annot=True)
    plt.xlabel(f"True Label \n Accuracy: {accuracy}")
    plt.ylabel("Predicted Label")

    filename = "conf_mse.jpg" if mse else "conf_ciou.jpg"
    plt.savefig(os.path.join("/content/drive/MyDrive/ECE 60146/HW5/Results", filename))

def draw_rectangles(image, bbox, predicted_bbox, label, predicted_label):
    # categories = {"pizza": 0, "bus": 1, "cat": 2}
    inverse_categories = {0: "pizza", 1: "bus", 2: "cat"}

    label = inverse_categories[int(label.item())]
    predicted_label = inverse_categories[int(predicted_label.item())]

    image = np.asarray(tvt.ToPILImage()(image))
    image = cv2.rectangle(image, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), color=(36, 255, 12), thickness=2)
    image = cv2.putText(image, label, (int(bbox[0]), int(bbox[1] - 10)), fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(36, 255, 12), thickness=2)

    image = cv2.rectangle(image, (int(predicted_bbox[0]), int(predicted_bbox[1])), (int(predicted_bbox[2]), int(predicted_bbox[3])), color=(255, 45, 12), thickness=2)
    image = cv2.putText(image, predicted_label, (int(predicted_bbox[0]), int(predicted_bbox[1] - 10)), fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(255, 45, 12), thickness=2)
    return image

def display_bbox(image_info, mse=True):
    fig, ax = plt.subplots(3, 3)
    row, col = 0, 0
    for idx, details in enumerate(image_info):
        image = details["Image"]
        original_label = details["Original Label"]
        predicted_label = details["Predicted Label"]
        original_bbox = details["Original Bbox"]
```

```
        predicted_bbox = details["Predicted Bbox"]

        image = draw_rectangles(image, original_bbox, predicted_bbox, original_label, predicted_label)
        ax[row, col].imshow(image)
        row = row+1 if not ((idx + 1) % 3) else row
        col = 0 if not((idx + 1) % 3) else col+1

        if(row == 3 and col == 0):
            break

    filename = "bbox_mse.jpg" if mse else "bbox_ciou.jpg"
    plt.savefig(os.path.join("/content/drive/MyDrive/ECE 60146/HW5/Results", filename))

def display_input_bbox(dataloader):
    inverse_categories = {0: "pizza", 1: "bus", 2: "cat"}
    fig, ax = plt.subplots(3, 3)
    row, col = 0, 0
    for batch_idx, (images, labels, bboxs) in enumerate(dataloader):
        for idx in range(len(labels)):
            if(col == 3):
                row += 1
                col = 0
            if(row == 3):
                break
            image, label, bbox = images[idx], inverse_categories[int(labels[idx])], bboxs[idx]
            image = np.asarray(tvt.ToPILImage()(image))
            image = cv2.rectangle(image, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), color=(36, 255, 12), thickness=2)
            image = cv2.putText(image, label, (int(bbox[0]), int(bbox[1] - 10)), fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(36, 255, 12), thickness=2)
            ax[row, col].imshow(image)
            col += 1
        if(row == 3):
            break

    plt.savefig(os.path.join("/content/drive/MyDrive/ECE 60146/HW5/Results", "inputs.jpg"))

def parser():
    parser = argparse.ArgumentParser(description="Object Detection and Localization")
    parser.add_argument("--epochs", type=int, help="Number of epochs", default=8)
    parser.add_argument("--mse", action="store_true", help="Choosing MSE Loss")
    args = parser.parse_args()
    return args


if __name__ == "__main__":
```

```
    args = parser()
    epochs = args.epochs
    regression = True if args.mse else False

    trainloader = get_training_dataloader()
    testloader = get_testing_dataloader()
    display_input_bbox(trainloader)

    model = HW5Net()
    label_loss, regression_loss = train(model, trainloader, epochs=epochs,
    mse=regression)
    plot_labeling_loss(label_loss, epochs=epochs)
    plot_regression_loss(regression_loss, epochs=epochs, mse=regression)

    confusion_matrix, accuracy, image_info = test(model, testloader, mse=
   regression)
    display_confusion_matrix(confusion_matrix, accuracy, mse=regression)
    display_bbox(image_info, mse=regression)
```

Listing 2: Training

3 Results

```
Number of layers: 172
```

Figure 2: Number of Layers in the Network

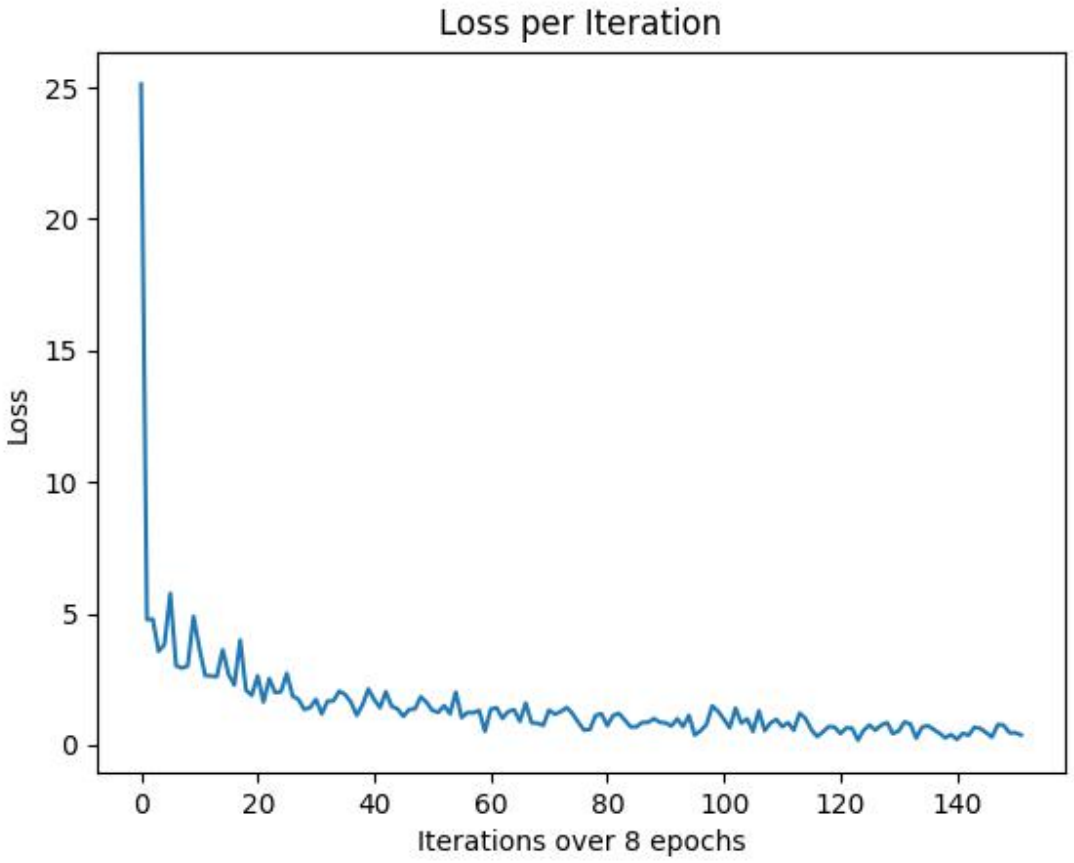

Figure 3: Cross Entropy Loss for Classification

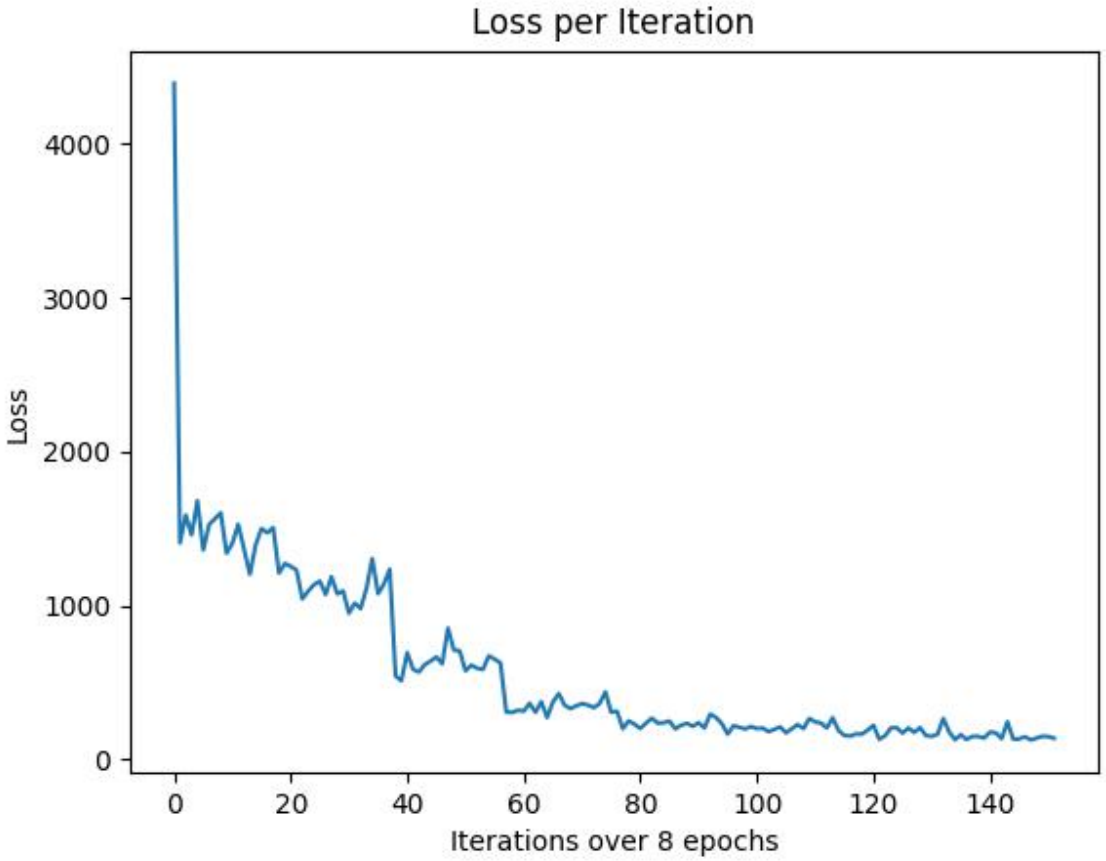

Figure 4: MSE Loss for Bounding Box

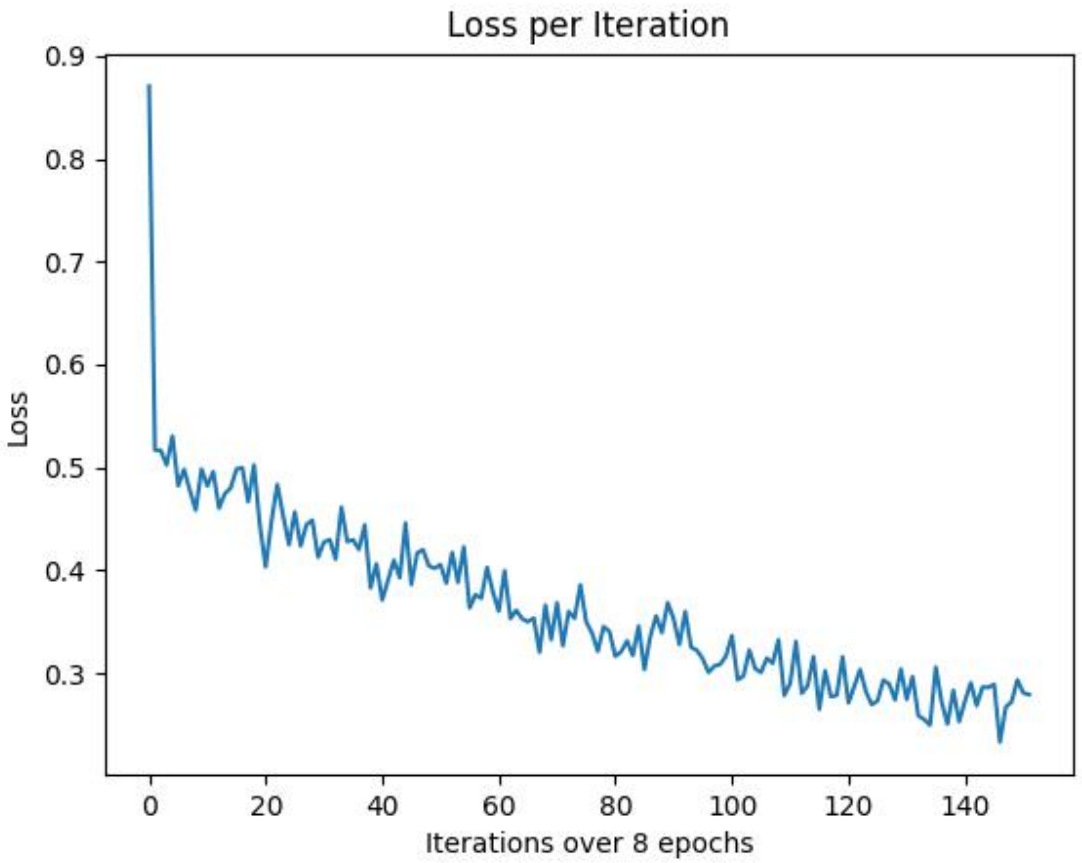

Figure 5: Complete IOU Loss

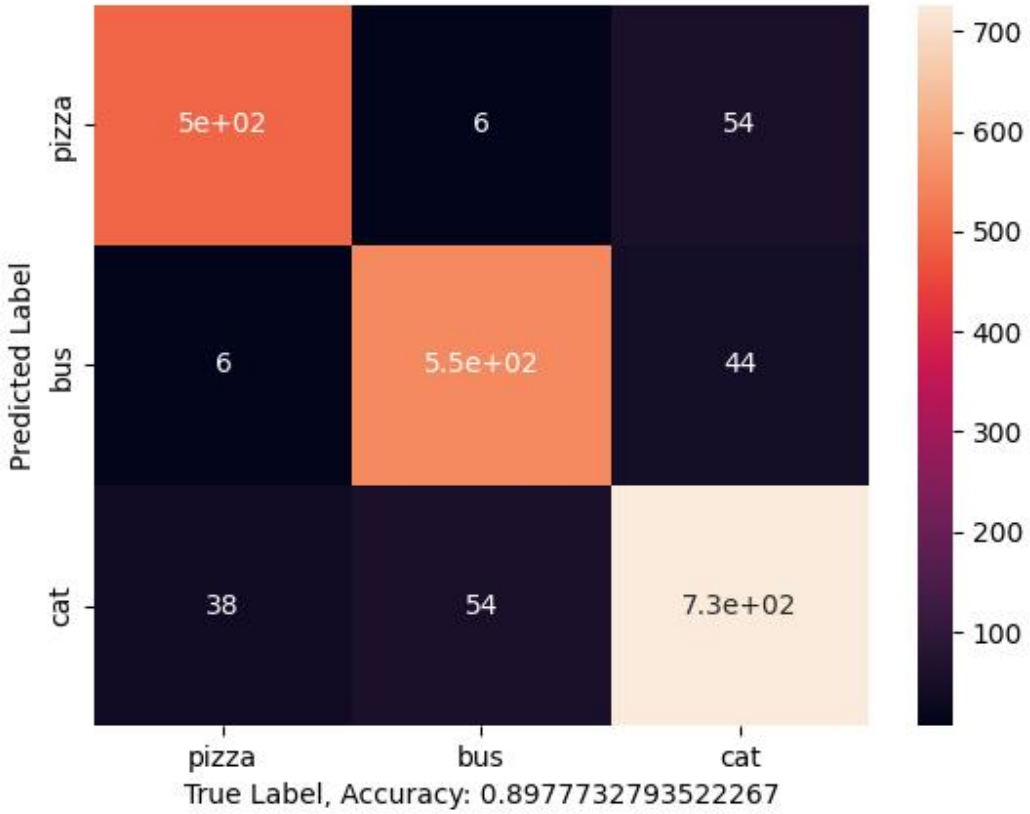

Figure 6: Confusion Matrix when using MSE Loss

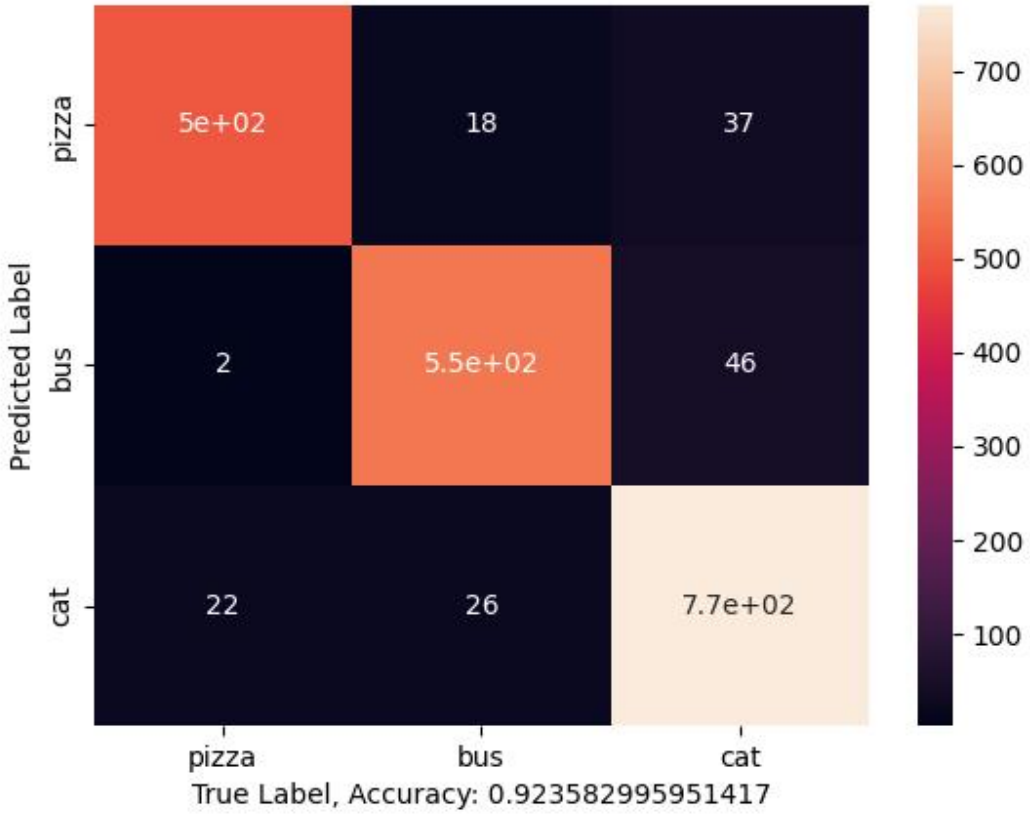

Figure 7: Confusion Matrix when using Complete IOU Loss

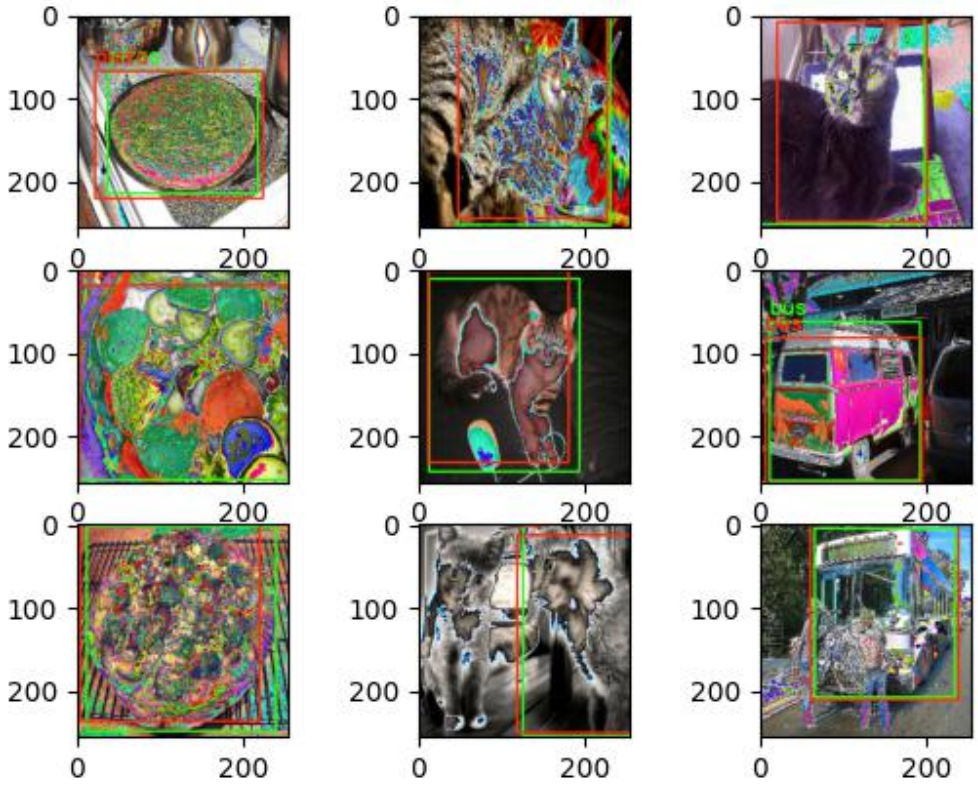

Figure 8: Bounding Box Predictions using MSE Loss

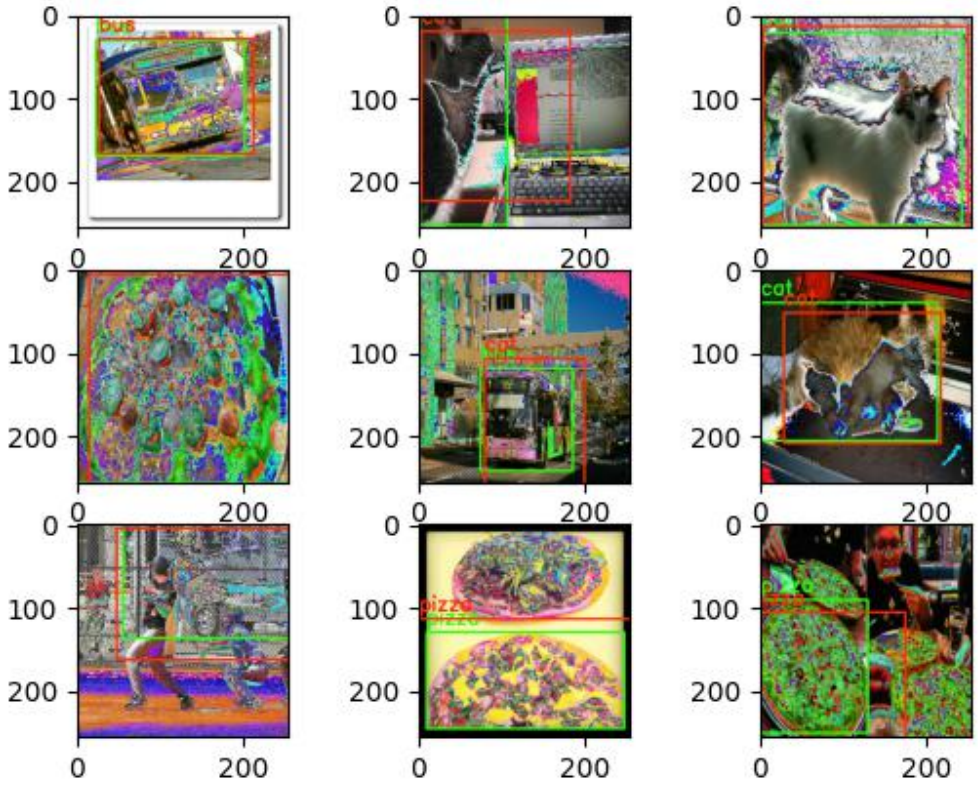

Figure 9: Bounding Box Predictions using Complete IOU Loss

4 Evaluation

This model consists of 172 layers. Using the MSE loss, the accuracy obtained was 89.7%, while using the Complete IoU loss gave an accuracy of 92.4%. In this particular situation, the Complete IoU loss function slightly outperformed the MSE loss function. In both cases, the bounding boxes are slightly off from the actual bounding box. Some enhancements that can be made to this model are:

- Adding more convolutional layers
- Revising the skip block network to something similar to the one found in Resnet
- Changing the learning rate
- Increasing the number of epochs