HW 4: CNN Image Categorization

David E. Farache, Email ID: dfarache@purdue.edu

March 7, 2023

1 Introduction

The goal of this homework is to create a pizza identifier with the implement of skip blocks within the network into a convolutional neural network (CNN). Skip-block block functions by adding the prior importance to the end of a network during backward propagation in order to increase the effect of the loss calculated. The evaluation of this networks must be done using MSE loss regression and Complete Box IOU loss.

2 Methodology

COCO was utilized to create an initial training set via the pycocotools.coco function, which would reduce to 3790 for training and 1987 for validation. The data-loader was based on HW4 with the addition of inter encoding based on the category and transform the image from a PIL image to a normalized tensor.

Across the course we have utilized CNN for predicting properties and are now expanding to ever deeper networks but these CNNs have an issue that with increasing depth there is degradation or a vanishing gradient. Skip-blocks resolve this issue by passing information from the initial layers to those deeper in the network, henceforth creating a "skip" or "short-cut". In this case our skip-block is a really a resnet block as the information is passed via matrix addition.

The CNN model was based on the DLStudio network with reduction of linear layer due to gpu space limitations. For this BatchNorm2d, the Skip-Block that was created, and Con2d for convoluational layers. The network had for loops that created layers of skip of depth 8.

For loss calculation, the classification is measured using Cross-Entropy loss while the bounding box is based upon the MSE regression loss.

$$Cross-Entropy = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{1}$$

$$MSE = \sum_{i=1}^{D} (x_i - y_i)^2 \tag{2}$$

For boundary box analysis, IoU is typically utilized which measure loss by taking 1 minus the area of overlap by the area of union. GIoU came about in order to grant an error even when there is no intersection so that the model could move into the proper direction and not keep guessing until it overlaps with the box. The issue was that although GIoU helped find the box, it had issue fitting.

Complete Box IoU was mode to combat this issue by introducing aspect ratio between bonding boxes so that the shape matches more so that being the right position.

$$CIoULoss = 1 - IoU + \frac{|C - B \cup B^{gt}}{|C|} + \frac{d^2}{c^2} + \alpha * v \tag{3}$$

3 Task 1: COCO Data-set Preparation

For task 1, the COCO data-set was used from train2014 and val2014 in order to reduce and create a training and validation for categories: pizza, cat, and bus. Images were selected based on whether there was a single dominant label with area greater than 200 x 200. Each image was turned in RGB and resized from their original height to 256 x 256. The bbox of the image was also re-scaled. A data-frame of filename, path, category, and bbox dimensions was saved for efficient processing.

```
# %% [markdown]
# # Library

# %%
%matplotlib inline
from pycocotools.coco import COCO

import numpy as np
import skimage.io as io
import skimage
import cv2
import pandas as pd

import matplotlib.pyplot as plt
import pylab
import random
from PIL import Image
pylab.rcParams[’figure.figsize’] = (8.0, 10.0)

# %% [markdown]
# # Display Images

# %%
# Based from skeleton code given
def display_random_image_with_bbox(new_image_path, filename, bbox, cat):
    file = new_image_path + ’/’ + filename
    image = Image.open(file)
    [x, y, w, h] = bbox
    fig, ax = plt.subplots(1,1)
    image = np.uint8(image)
    image = cv2.rectangle(image, (int(x), int(y)), (int(x + w), int(y + h)), color=(36, 255, 12), thickness=2)
    image = cv2.putText(image, cat, (int(x), int(y - 10)), fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(36, 255, 12), thickness=2)

    ax.imshow(image)
    ax.set_axis_off()
    plt.axis("tight")
    plt.show()

    fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(10,10), dpi = 150)
    axs = axs.flatten()
    axs_count = 0

# %%
def display_images(saveForPlotting):
    fig, ax = plt.subplots(3, 3)
    row, col = 0, 0

    #print(len(saveForPlotting))
    for arr in saveForPlotting:
        file = arr[0] + '/' + arr[1]
        image = Image.open(file)

        [x, y, w, h] = arr[2]
        image = np.uint8(image)
        image = cv2.rectangle(image, (int(x), int(y)), (int(x + w), int(y + h)), color=(36, 255, 12), thickness=2)
        image = cv2.putText(image, arr[3], (int(x), int(y - 10)), fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(36, 255, 12), thickness=2)
        ax[row, col].imshow(image)

        # Increment through row
        col += 1

        if col == 3:
            col = 0
            row += 1

    fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(10,10), dpi = 150)
    plt.show()
    axs = axs.flatten()

# %% [markdown]
# # Adjust and Change Images

# %%
def resizeBBOX(bbox, startWidth, startHeight, resize):
        x_scale = resize / startWidth
        y_scale = resize / startHeight
        bboxResize = np.zeros(4)

        bboxResize[0] = int(x_scale * bbox[0])
        bboxResize[1] = int(y_scale * bbox[1])
        bboxResize[2] = int(x_scale * bbox[2])
        bboxResize[3] = int(y_scale * bbox[3])

        return bboxResize.tolist()

# %%
def resizeAndRepairImage(start_image_path, new_image_path, filename, bbox, width, height, resize):
        img = Image.open(start_image_path + '/' + filename)
        width, height = img.size

        image = img.convert(mode="RGB")

        img = image.resize((resize, resize), Image.BOX)
        img.save(new_image_path + '/' + filename)

# %%
def saveAsDataFrame(ids, cats, filepaths, x1, y1, width, height, type):
    # Containers for data to go in training label csv
    columns = ["id", "category", "filepath", "x1", "y1" "width", "height"]
    dataFrame = pd.DataFrame(columns=columns)
    dataFrame["id"] = ids
    dataFrame["category"] = cats
    dataFrame["filepath"] = filepaths
    dataFrame["x1"] = x1
    dataFrame["y1"] = y1
    dataFrame["width"] = width
    dataFrame["height"] = height
    dataFrame.to_csv("{0}_labels.csv".format(type))


# %% [markdown]
# # Choose Images Within Parameters

# %%
# Get Random Images from set
def ImageSelection(start_image_path, new_image_path, cocoObj, class_list, type, spotSave):
    # Save File Location List
    saveImportant = []
    saveForPlotting = []

    # Save image info
    ids = []
    cats = []
    filepaths = []
    x1 = []
    y1 = []
    widthSave = []
    heightSave = []

    for cat in class_list:
        # get all images containing given categories
        catIds = cocoObj.getCatIds(catNms=[cat]) # Get ids from annotations
        imgIds = cocoObj.getImgIds(catIds=catIds ) # Load images ids of chosen annotations ids
        img = cocoObj.loadImgs(ids=imgIds) # Get images
        numPlots = 0 # Make X number plots


        #Loop per image
        for idx, images  in enumerate(img):
            annIds = cocoObj.getAnnIds(imgIds=images['id'], catIds=catIds, iscrowd=False) # Get dictionary value
            anns = cocoObj.loadAnns(annIds) # Get annotations
            domObj = 0 # Amount of Dominante Obj
            spot = 0
            for jdx, ann in enumerate(anns):
                if ann['area'] > 200 * 200: #check if means parameter of dominate obj
                    domObj += 1
```

```python
                    spot = jdx


            if domObj == 1: # only one dom allowed
                # Resize Images
                width, height = images['width'], images['height']
                boxDict = []
                #for ann in anns:
                bboxResize = resizeBBOX(anns[jdx]['bbox'], int(width), int(height), resize=256) #Adjust Box
                resizeAndRepairImage(start_image_path, new_image_path, images['file_name'], anns[jdx]['bbox'], width, height, resize=256) # Save Adjust Image

                ids.append(images['id'])
                cats.append(cat)
                filepaths.append("{0}/{1}".format(spotSave,images['file_name']))
                x1.append(bboxResize[0])
                y1.append(bboxResize[1])
                widthSave.append(bboxResize[2])
                heightSave.append(bboxResize[3])

                if numPlots < 3:
                    #display_random_image_with_bbox(new_image_path,images['file_name'], bboxResize, cat)
                    saveForPlotting.append([new_image_path, images['file_name'], bboxResize, cat])
                    print(saveForPlotting)
                    numPlots += 1

    display_images(saveForPlotting)
    saveAsDataFrame(ids, cats, filepaths, x1, y1, widthSave, heightSave, type)

# %%
# Input
train_json = '/Users/davidfarache/Documents/ECE60146/HW5/annotations/instances_train2014.json'
val_json = '/Users/davidfarache/Documents/ECE60146/HW5/annotations/instances_val2014.json'

train_path = '/Users/davidfarache/Documents/ECE60146/HW5/train2014'
train_data_path = '/Users/davidfarache/Documents/ECE60146/HW5/trainingData'

val_path = '/Users/davidfarache/Documents/ECE60146/HW5/val2014'
val_data_path = '/Users/davidfarache/Documents/ECE60146/HW5/valData'

trainSaveSpot = 'trainingData'
valSaveSpot = 'valData'

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

# %%
cocoTrain = COCO(train_json)
cocoVal = COCO(val_json)

ImageSelection(train_path, train_data_path, cocoTrain, class_list, 'train', trainSaveSpot)
ImageSelection(val_path, val_data_path, cocoVal, class_list, 'val', valSaveSpot)

# %%
```

Listing 1: *Creating COCO data-set*

A sample of the images select has been placed below, which can bee see to have shrunk and had their boundry box rescaled within them:

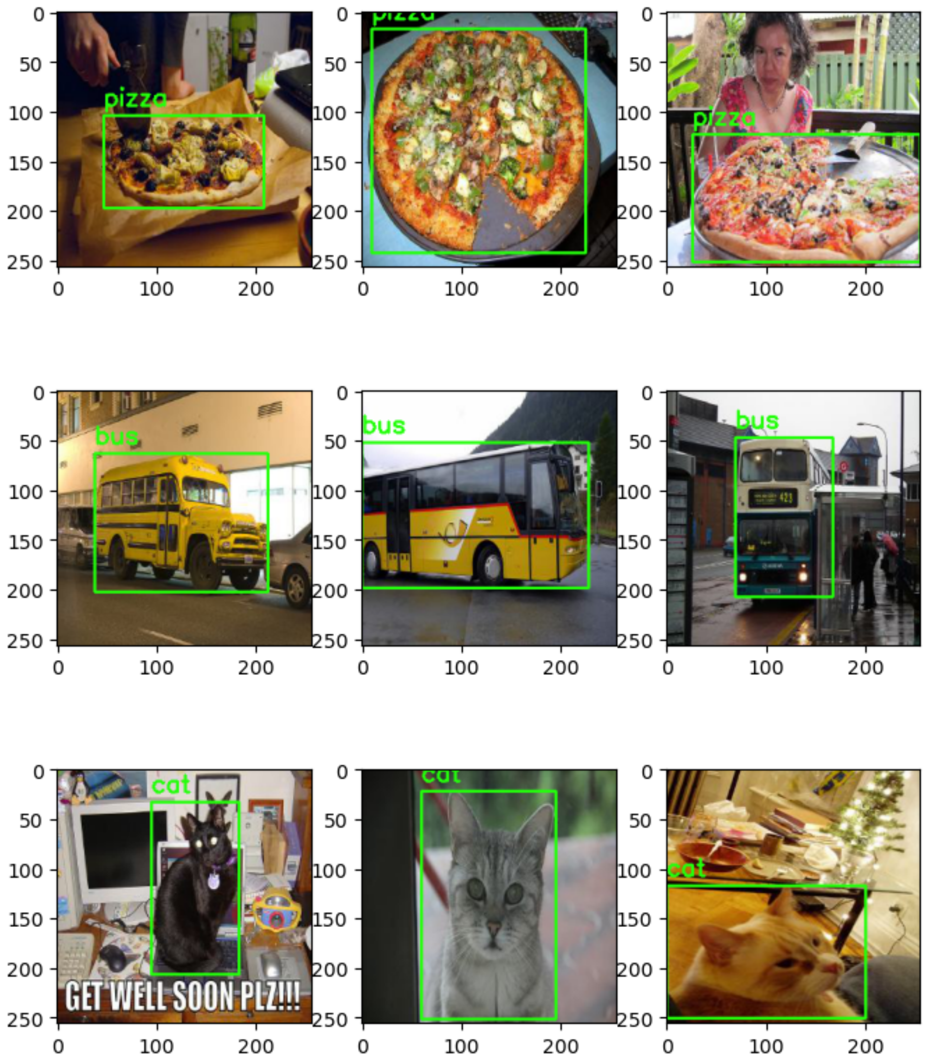


Figure 1: *Selection of Training Set*

4 Task 2: Create Skip-Connection Block and CNN

The network created utilizes a training and validation set created from COCO. The following code has the data-set which stores boundary box, normalized image, and the image label. The depth was set to 8, with learning rate of 1e-4, trained for 7 epochs, and running average loss value is taken per 100 iterations. Two training functions were created one that performed Cross-Entropy and MSE loss (trainMSERegression) and the other using Cross-Entropy and CIoU (trainCompleteBoxIOU). The Adam optimizer was selected for both. The testing was done the same for both.

```python
# %%
# Import Libraries
import numpy as np

# PyTorch
import torch
import torchvision.transforms as tvt
import torch.utils.data
import torch.nn as nn
import torch.nn.functional as F
from torchvision.ops import complete_box_iou_loss

# Data Processing
from PIL import Image
import os
import cv2
import pandas as pd

# Plotting
import matplotlib.pyplot as plt
import seaborn as sns

# GLOBAL VARIABLES
device = ’cuda’ if torch.cuda.is_available() else ’cpu’
device = torch.device(device)

# %% [markdown]
# # Save Annotations into Dictionary

# %%
def ImageProcessing(images):
    # Get dir
    image_dir = images["filepath"]

    # Get inputs
    image = Image.open(image_dir)
    bbox = [images["x1"], images["y1"], images["width"], images["height"]]

    # Normalize image
    toTensor = tvt.ToTensor()(image)
    toNormalize = tvt.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))(toTensor)
    return toNormalize, bbox

# %%
# Create Dataset
class MyDataset(torch.utils.data.Dataset):
    # Obtain meta information (e.g. list of file names)
    def __init__(self, imagesDataFrame, class_list):
        super().__init__()
        self.imagesDataFrame = imagesDataFrame
        self.catEncoding = { # Set Integer Values for Cat
            "bus" : 0,
            "cat" : 1,
            "pizza" : 2
        }

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

    def __getitem__(self, i):
        imagesDataFrame = self.imagesDataFrame.iloc[i]
        normalImage, bbox = ImageProcessing(imagesDataFrame)
        label = self.catEncoding[imagesDataFrame["category"]]

        # Fix box of resize
        width, height = bbox[2], bbox[3]
        bboxAdjust = [bbox[0], bbox[1], width, height]
        bboxTensor = torch.tensor(bboxAdjust, dtype=torch.float)
        return normalImage, label, bboxTensor

# %% [markdown]
# # Skip Block

# %%
# Based on DLStudio SkipBlock Code
# Based off following https://blog.paperspace.com/writing-resnet-from-scratch-in-pytorch/
class Block(nn.Module):
      def __init__(self, in_ch, out_ch, downsample=False, skip_connections=True):
          super(Block, self).__init__()
          self.downsample = downsample
          self.in_ch = in_ch
          self.out_ch = out_ch

          self.skip_connections = skip_connections

          self.conv1 = nn.Sequential(
                        nn.Conv2d(in_ch, out_ch, kernel_size = 3, stride = 1, padding = 1),
                        nn.BatchNorm2d(out_ch),
                        nn.ReLU())
          self.conv2 = nn.Sequential(
                            nn.Conv2d(out_ch, out_ch, kernel_size = 3, stride = 1, padding = 1),
                            nn.BatchNorm2d(out_ch))
          self.relu = nn.ReLU()

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

      def forward(self, x):
          residual = x
          out = self.conv1(x)

          if self.in_ch == self.out_ch:
              out = self.conv1(out)

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

          if self.skip_connections:
              if self.in_ch == self.out_ch:
                  out = out + residual
              else:
                  # Assuming equivalent dimensions which this dataset fits
                  firstSection = out[:,:self.in_ch,:,:]
                  secondSection = out[:,self.in_ch:,:,:]

                  out = torch.cat((firstSection + residual, secondSection + residual), dim=1)

          return out

# %% [markdown]
# # CNN

# %%
# CNN model # Inspired by DLStudio LOADnet2 && notes from class
class CNN(nn.Module):
    def __init__(self, skip_connections=True, depth=8):
        super(CNN, self).__init__()
        self.skip_connections = skip_connections
        self.depth = depth // 2

        # Create base layers
        self.conv = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Classification
        self.bn1 = nn.BatchNorm2d(num_features=64)
        self.bn2 = nn.BatchNorm2d(num_features=128)

        # Create Depth layers 64
        self.skip64_arr = nn.ModuleList()
        for idx in range(self.depth):
            self.skip64_arr.append(Block(in_ch=64, out_ch=64, skip_connections=self.skip_connections))

        self.skip64ds = Block(in_ch=64, out_ch=64, downsample=True, skip_connections=self.skip_connections)
        self.skip64to128 = Block(in_ch=64, out_ch=128, skip_connections=self.skip_connections)

        # Create depth layer 128
        self.skip128_arr = nn.ModuleList()
        for idx in range(self.depth):
            self.skip128_arr.append(Block(in_ch=128, out_ch=128, skip_connections=self.skip_connections))
        self.skip128ds = Block(in_ch=128, out_ch=128, downsample=True, skip_connections=self.skip_connections)

        # Linear Layers #Limited from original file do to lack of memory space
        self.fc1 = nn.Linear(in_features=32*32*128, out_features=3) # 3 categories

        ## For Regression
        self.conv_sequential = nn.Sequential(nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
```

```python
                                        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_sequential = nn.Sequential(nn.Linear(in_features=128*128*64, out_features=4)) # 4 dimensions bbox

    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)
        cls = cls.view(-1, 32 * 32 * 128)
        cls = self.fc1(cls)

        # Regression for BBox
        bbox = self.conv_sequential(x)
        bbox = bbox.view(x.size(0), -1)
        bbox = self.fc_sequential(bbox)

        return cls, bbox

# %% [markdown]
# # Training

# %%
############################## TRAINING ##############################
# Inspired by DLStudio run_code_for_training_with_CrossEntropy_and_MSE_Losses && notes from class
def trainMSERegression(model, trainDataLoader, runn_avg_size, epochs, lr=1e-4, betas=(0.9, 0.99)):
    model = model.to(device)
    num_layers = len(list(model.parameters()))
    print(f"Number of layers: {0}".format(num_layers))

    classification_criterion = nn.CrossEntropyLoss()
    regression_criterion = nn.MSELoss()

    optimizer = torch.optim.Adam(model.parameters(), lr=lr, betas=betas)
    labeling_loss_running_avg = []
    regression_loss_running_avg = []
    for epoch in range(1, epochs+1):
        running_loss_labeling = 0.0
        running_loss_regression = 0.0
        for batch_idx, (imgTensor, labels, bbox) in enumerate(trainDataLoader):
            imgTensor = imgTensor.to(device)
            labels = labels.to(device)
            bbox = bbox.to(device)

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

            loss_label = classification_criterion(output_label, labels)
            running_loss_labeling += loss_label.item()
            loss_label.backward(retain_graph=True)

            loss_bbox = regression_criterion(output_bbox, bbox)
            running_loss_regression += loss_bbox.item()
            loss_bbox.backward()

            optimizer.step()

            if(batch_idx % runn_avg_size == (runn_avg_size - 1)):
                labeling_loss_running_avg.append(running_loss_labeling / float(runn_avg_size))
                regression_loss_running_avg.append(running_loss_regression / float(runn_avg_size))

                running_loss_labeling = 0.0
                running_loss_regression = 0.0

    return labeling_loss_running_avg, regression_loss_running_avg

# %%
############################## TRAINING ##############################
# Inspired by DLStudio run_code_for_training_with_CrossEntropy_and_MSE_Losses && notes from class
def trainCompleteBoxIOU(model, trainDataLoader, runn_avg_size, epochs, lr=1e-4, betas=(0.9, 0.99)):
    model = model.to(device)
    num_layers = len(list(model.parameters()))
    print(f"Number of layers: {0}".format(num_layers))

    classification_criterion = nn.CrossEntropyLoss()
    regression_criterion = complete_box_iou_loss

    optimizer = torch.optim.Adam(model.parameters(), lr=lr, betas=betas)
    labeling_loss_running_avg = []
    regression_loss_running_avg = []

    for epoch in range(1, epochs+1):
        running_loss_labeling = 0.0
        running_loss_regression = 0.0
        for batch_idx, (imgTensor, labels, bbox) in enumerate(trainDataLoader):
            imgTensor = imgTensor.to(device)
            labels = labels.to(device)
            bbox = bbox.to(device)

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

            loss_label = classification_criterion(output_label, labels)
            loss_label.backward(retain_graph=True)
            running_loss_labeling += loss_label.item()

            loss_bbox = regression_criterion(output_bbox, bbox, reduction = "mean")
            loss_bbox.backward()
            running_loss_regression += loss_bbox.item()

            optimizer.step()

            if(batch_idx % runn_avg_size == (runn_avg_size - 1)):
                labeling_loss_running_avg.append(running_loss_labeling / float(runn_avg_size))
                regression_loss_running_avg.append(running_loss_regression / float(runn_avg_size))

                running_loss_labeling = 0.0
                running_loss_regression = 0.0

    return labeling_loss_running_avg, regression_loss_running_avg

# %% [markdown]
# # Testing

# %%
def test(model, testDataLoader, class_list):
    model = model.to(device)
    confusion_matrix = np.zeros((len(class_list), len(class_list)))
    image_data = []

    with torch.no_grad():
        for imgTensor, labels, bbox in testDataLoader:
            # Set to Device
            imgTensor = imgTensor.to(device)
            labels = labels.to(device)
            bbox = bbox.to(device)

            # Set Outputs
            outputs = model(imgTensor)
            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(imgTensor, labels, predicted, bbox, output_bbox):
```

```python
                image_data.append([img, original_label, predicted_label, original_bbox, predicted_bbox])

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

Listing 2: *Skip-block CNN training and testing code*

5 Task 3: Validation

For validation a loss plot vs iteration was made for the Cross-Entropy, MSE, and CIoU loss. A confusion matrix was further created to compare accuracy based on MSE and CIoU loss.

```python
# %%
def plotRegressionLosses(regression, lossType):
    figure = plt.figure(1)

    plt.plot(range(len(regression)), regression, label="Loss")

    plt.xlabel("Iterations")
    plt.ylabel("Loss")
    plt.legend(loc="lower right")

    plt.savefig("lossvsiter_regress{0}.jpg".format(lossType))

# %%
def plotLabelLosses(labeling, lossType):
    figure = plt.figure(2)

    plt.plot(range(len(labeling)), labeling, label="Loss")

    plt.xlabel("Iterations")
    plt.ylabel("Loss")
    plt.legend(loc="lower right")

    plt.savefig("lossvsiter_class{0}.jpg".format(lossType))

# %%
def plotConfusionMatrix(conf, accuracy, class_list, lossType):
    figure = plt.figure(3)
    sns.heatmap(conf, xticklabels=class_list, yticklabels=class_list, annot=True)
    plt.xlabel("True Label: Accuracy {0}".format(accuracy))
    plt.ylabel("Predicted Label")
    plt.savefig("confMatrix{0}.jpg".format(lossType))

# %%
def plotImages(image_data, lossType):
    fig, ax = plt.subplots(3, 3)
    row, col = 0, 0

    for idx, arr in enumerate(image_data):
        image = tvt.ToPILImage()(arr[0])

        invert_categories = { # Set Integer Values for Cat
            0 : "bus",
            1 : "cat",
            2 : "pizza"
        }

        original_cat = invert_categories[arr[1].item()]
        predicted_cat = invert_categories[arr[2].item()]
        original_bbox = arr[3]
        predicted_bbox = arr[4]

        # Original bbox
        #print(original_bbox)
        [x, y, w, h] = original_bbox
        image = np.uint8(image)
        image = cv2.rectangle(image, (int(x), int(y)), (int(x + w), int(y + h)), color=(36, 255, 12), thickness=2)
        image = cv2.putText(image, original_cat, (int(x), int(y - 10)), fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(36, 255, 12), thickness=2)

        [x, y, w, h] = predicted_bbox
        # image = np.uint8(image)
        image = cv2.rectangle(image, (int(x), int(y)), (int(x + w), int(y + h)), color=(255, 36, 12), thickness=2)
        image = cv2.putText(image, predicted_cat, (int(x), int(y - 10)), fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=0.8, color=(255, 36, 12), thickness=2)

        print(row, col)
        ax[row, col].imshow(image)

        # Increment through row
        col += 1
        if col == 3:
            col = 0
            row += 1

        if idx > 7:
            break

    # fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(10,10), dpi = 150)
    # plt.show()
    # axs = axs.flatten()
    plt.savefig("predvsorginal_bbox{0}.jpg".format(lossType))
```

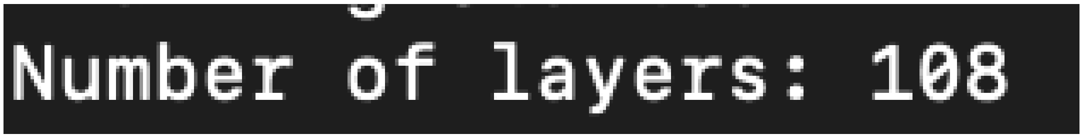
Listing 3: *Plotting and Validation Code*

```python
# Images saved directory
train_data_path = 'trainingData'
val_data_path = 'valData'

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

trainDataFrame = pd.read_csv('train_labels.csv')
valDataFrame = pd.read_csv('val_labels.csv')

# %%
trainDataset = MyDataset(trainDataFrame, class_list)
valDataset = MyDataset(valDataFrame, class_list)
```

```python

trainDataloader = torch.utils.data.DataLoader(trainDataset, batch_size=2, num_workers=2, shuffle=True)
valDataloader = torch.utils.data.DataLoader(valDataset, batch_size=2, num_workers=2, shuffle=True)

# %%
# Save networks
net = CNN()

# Train Networks
epochs=7
labelMSELoss, regressMSELosses = trainMSERegression(net, trainDataloader,  100, epochs=epochs, lr=1e-4, betas=(0.9, 0.99))
lossType = 'MSELoss'
plotRegressionLosses(regressMSELosses, lossType)
plotLabelLosses(labelMSELoss, lossType)

conf, acc, image_data = test(net, valDataloader, class_list)
plotConfusionMatrix(conf, acc, class_list, lossType)
plotImages(image_data, lossType)

labelCBIOULoss, regressCBIOULosses = trainCompleteBoxIOU(net, trainDataloader, 100, epochs=epochs, lr=1e-4, betas=(0.9, 0.99))
lossType = "CIoULoss"
plotRegressionLosses(regressCBIOULosses, lossType)
plotLabelLosses(labelCBIOULoss, lossType)

conf, acc, image_data = test(net, valDataloader, class_list)
plotConfusionMatrix(conf, acc, class_list, lossType)
plotImages(image_data, lossType)
```

Listing 4: *Main for running all prior code*

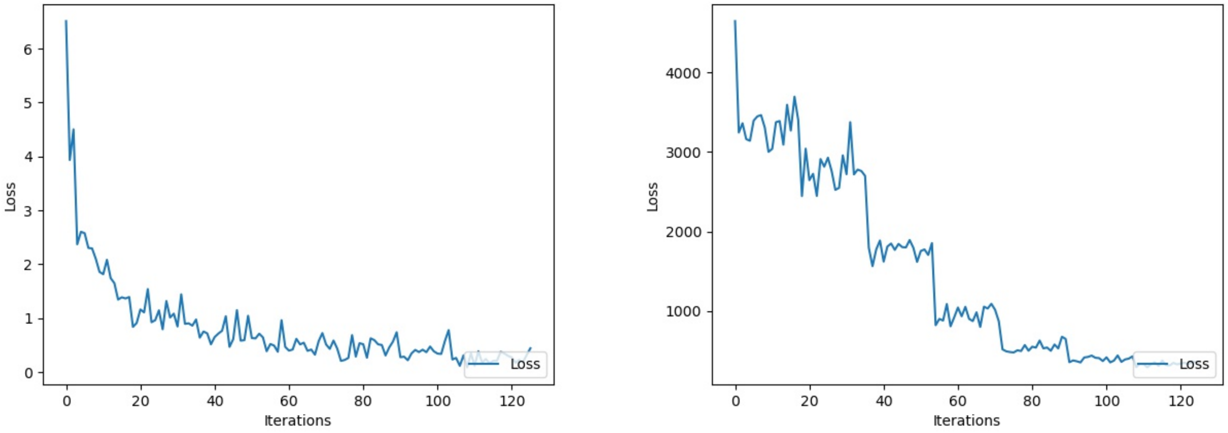
The following images are the results and validation for the model created in the prior code. It will be shown that all models met criteria of high enough accuracy and that the number of layer the system surpassed is greater than 50 as seen in Figure 2.



```
Number of layers: 108
```

Figure 2: *Layer in network*

The model was made up of 108 layers that, when trained with MSE, granted a accuracy of 0.81 and, when trained with CIoU, granted a accuracy of 0.8. This indicates that MSE Regression Loss outperforms CIoU but this could be due to the effects of one being run a cluster with gpu access and another run on google colab. Based on the actual figure 6 and 7 one can see that MSE performs better than the other in terms of fitting boxes which seems to be due to a confusion another object in the image. The network could be improved by simply adding more layers, running for more epochs, or shifting the learning as with any model. A unique solution would be changing the skip-block made to resemble more advance methods that do not make the same assumptions that our does.



(a) *Cross-Entropy Loss Per Iteration*

(b) *MSE Regression Loss*

Figure 3: *Training Loss for Cross-Entropy and MSE loss*

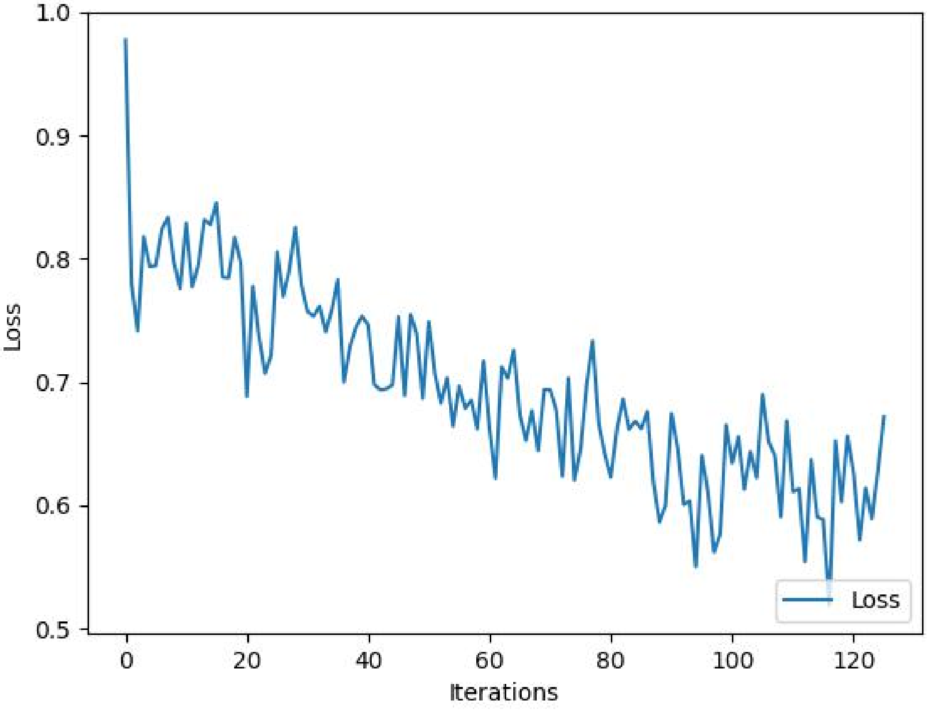
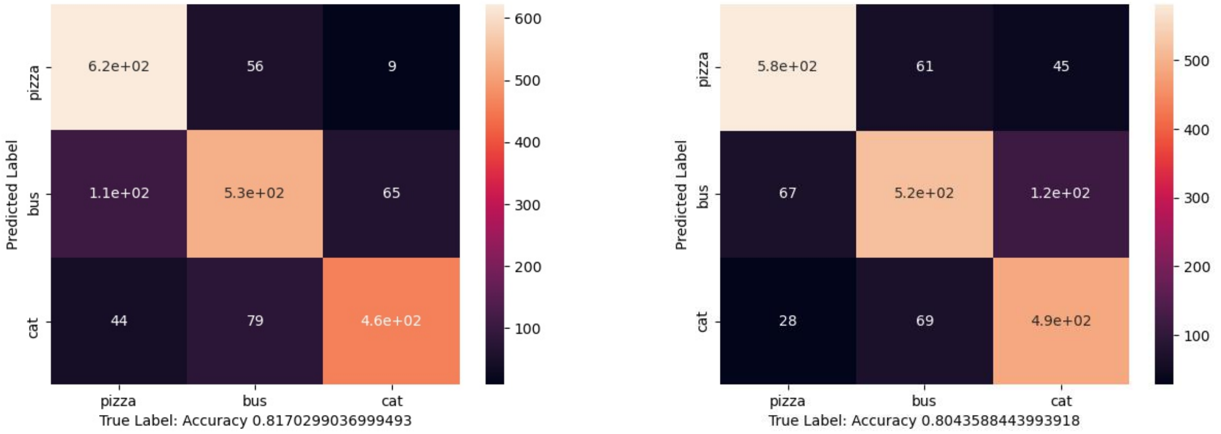


Figure 4: *Complete Box IoU Loss*



(a) *MSE Regression Loss*

(b) *Complete Box IoU*

Figure 5: *Confusion Matrixes*

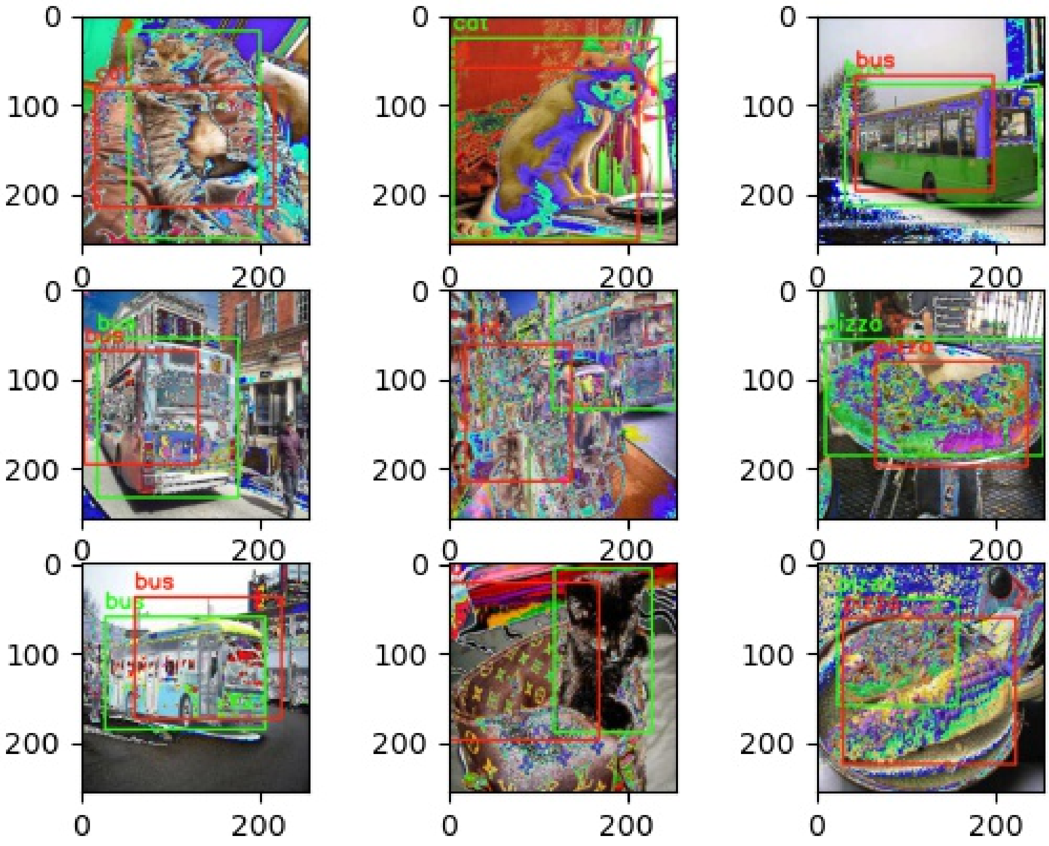


Figure 6: *MSE Regression Loss bounding box results*

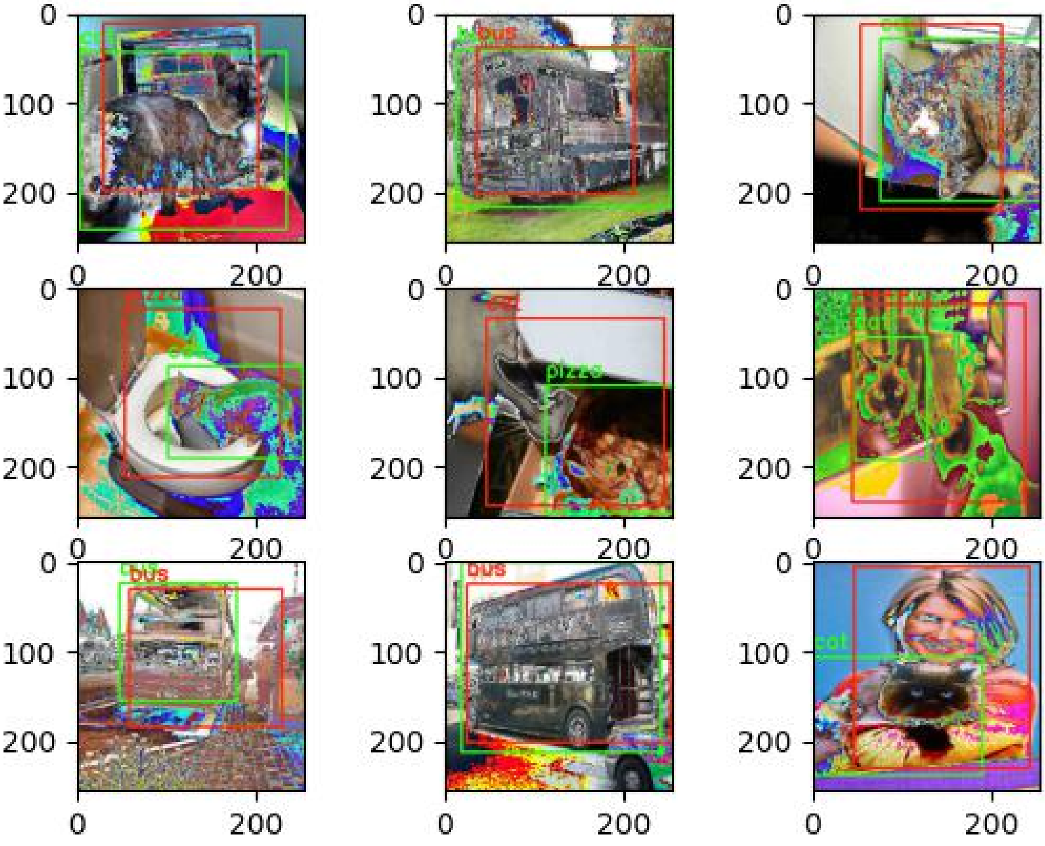


Figure 7: *CIoU Regression Loss bounding box results*

6 Lessons learned

From this assignment, I learned about boundary boxes, skip-block implementation, and CIoU.