BME 646/ ECE695DL: Homework 5

Varun Aggarwal

March 8, 2022

1 Introduction

This project was aimed at implementing COCO dataloader and creating a custom CNN with skip blocks. In addition, training and validation procedure has to be demonstrated with the new network.

2 Methodology

The assignment can be divided into four tasks.

1. Creating Skip Block
2. Setting up COCO Dataset
3. Setting up training function
4. Setting up testing function

Task 1

A skip was created by taking inspiration from Inception Network and ResNet. The Skipblock is shown in figure 1. The Skipblock has two independent paths and a skip connection. Overall, the network is composed of multiple such skipblocks determined by a variable called depth. At half the depth, the input is down sampled by using convolution layer in half.

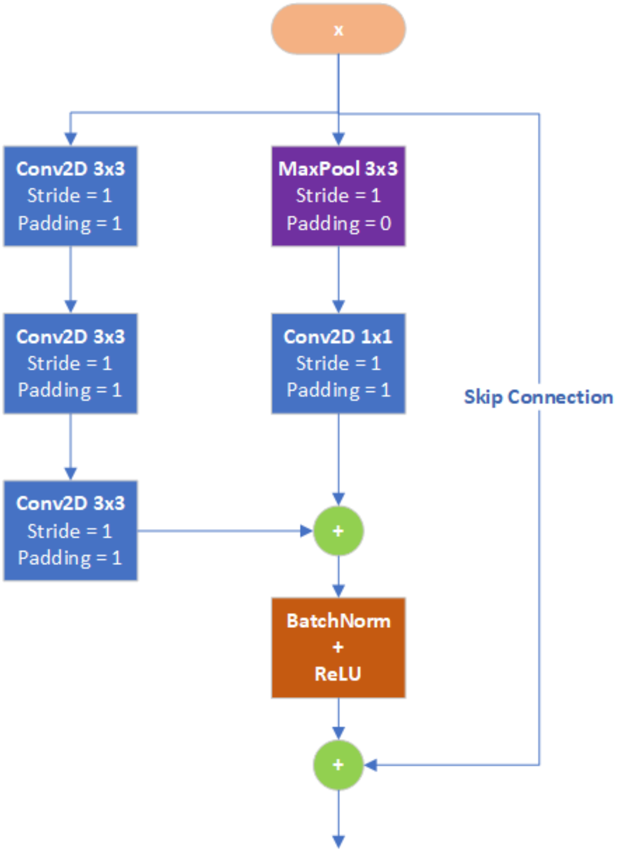


Figure 1: Skip Block for the CNN

Overall,the network consists of a network for initial processing of the image batch, followed by two branches. The first branch is used for classification while the other is used for the regression. THe network is given in Figure 2.

Task 2

COCO dataset is loaded using pycocotools library. This makes handling COCO dataset easy. The images are downloaded to disk during first run and during subsequent runs, if the file exists on the disk, download is skipped. In addition, the bounding box coordinates are also extracted for each image.

First, for each image, it is determined if the largest bounding box category is equal to image category. Next, the image is resized along with the bounding box coordinate. In

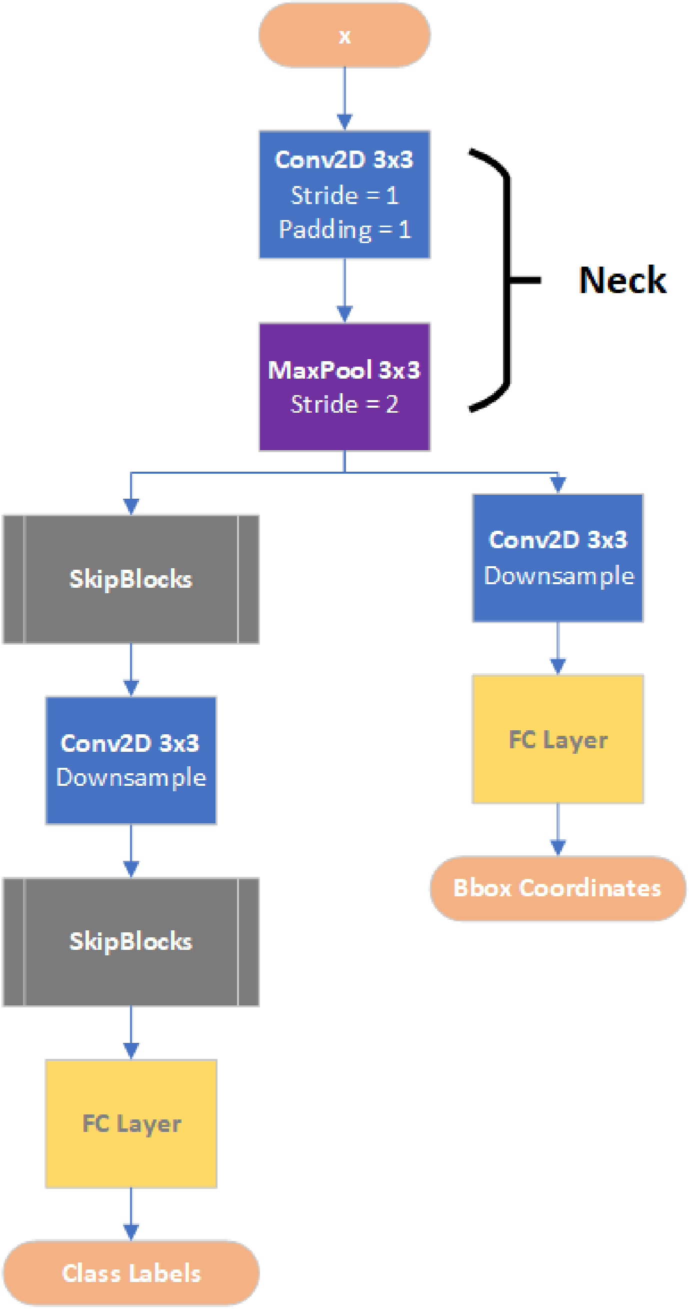


Figure 2: Complete Network

addition, bounding box coordinates are normalized between 0 and 1 using custom normalize function created in utils.py Finally, pickle is used to dump the collection of images and their corresponding bounding boxes for faster load times during subsequent testing.

Task 3

Training loop code is modified from DLStudio. Not a lot of changes had to made. Primarily, normalization and un-normalization was incorporated for correctly displaying the prediction and ground truth boxes. Overall, both classification and regression loss decreased with each epoch. Three classes were selected as a proof of concept. The classes were cat, train and airplane. The loss curves are given in Figure 3.

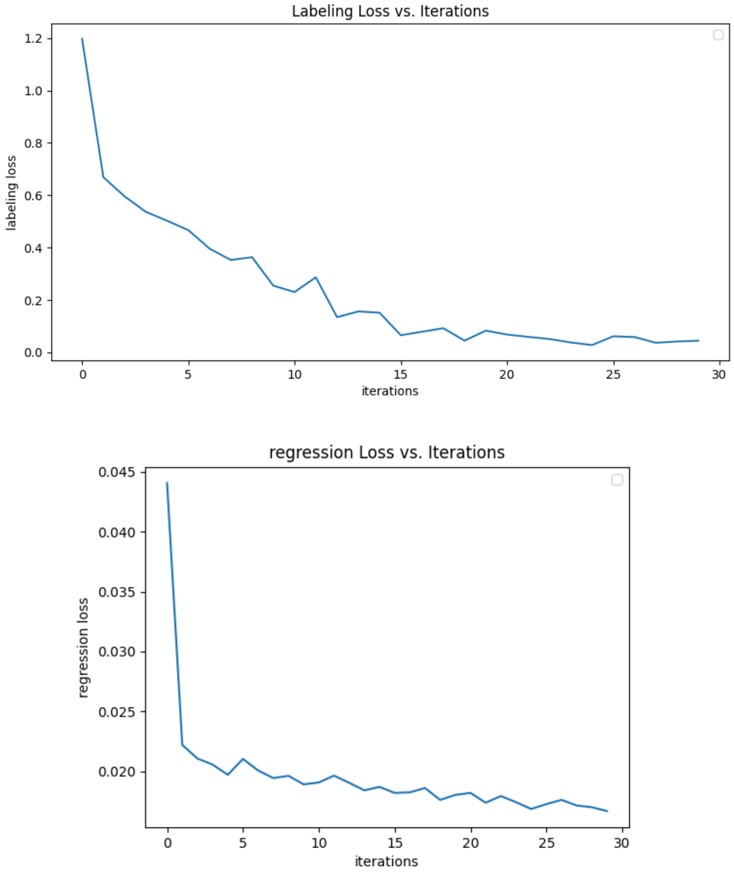
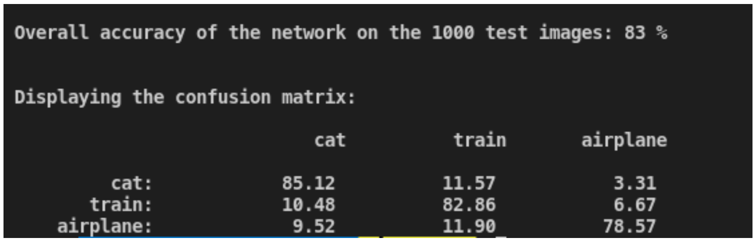


Figure 3: Classification and Regression Loss Curves

Task 4

Testing loop code is modified from DLStudio. Again, not a lot of changes had to made. Primarily, normalization and un-normalization was incorporated for correctly displaying the prediction and ground truth boxes. Overall classification accuracy was close to 82%. The confusion matrix is given in Figure 4. Additionally, an example of prediction on the image is given in Figure 5.



```
Overall accuracy of the network on the 1000 test images: 83 %


Displaying the confusion matrix:

                             cat           train        airplane

        cat:               85.12           11.57            3.31
      train:               10.48           82.86            6.67
   airplane:                9.52           11.90           78.57
```

Figure 4: Overall Accuracy and Confusion Matrix for three classes

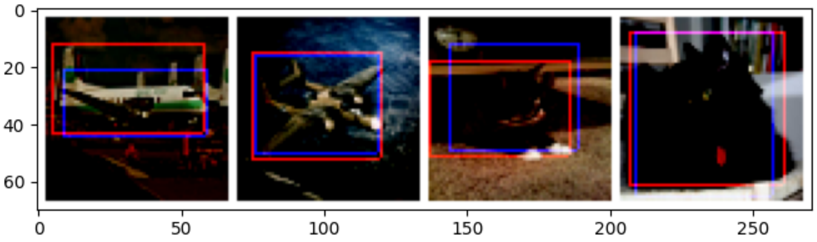


Figure 5: Prediction and Ground Truth

3 Implementation

CODE-hw05_training.py

```
import copy, time
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as tvt
import torch.optim as optim
from pycocotools.coco import COCO
import os, sys, logging
import matplotlib.pyplot as plt
# USER imports
sys.path.append("/home/varun/work/courses/why2learn/hw/DLStudio-2.1.6/")
from DLStudio import *

sys.path.append("/home/varun/work/courses/why2learn/hw/DLStudio-2.1.6/
    ↪ Examples")
from model import pneumaNet

from dataloader import CocoDetection
import utils

class tool(DLStudio):
    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)

    class fearInoculum(DLStudio.DetectAndLocalize):
        def __init__(self, dl_studio, dataserver_train=None,dataserver_test=
            ↪ None,dataset_file_train=None,dataset_file_test=None,):
            super().__init__(dl_studio, dataserver_train,dataserver_test,
                ↪ dataset_file_train,dataset_file_test,)
            self.dl_studio = dl_studio

        def run_code_for_training_with_CrossEntropy_and_MSE_Losses(self, net
            ↪ ):
            # create files for saving trainng logs
            filename_for_out1 = "performance_numbers_" + str(self.dl_studio.
                ↪ epochs) + "label.txt"
            filename_for_out2 = "performance_numbers_" + str(self.dl_studio.
                ↪ epochs) + "regres.txt"
            FILE1 = open(filename_for_out1, ’w’)
            FILE2 = open(filename_for_out2, ’w’)

            # copy the model
            net = copy.deepcopy(net)
            net = net.to(self.dl_studio.device)

            # setup criterions for backprop
            criterion1 = nn.CrossEntropyLoss()
```

```
criterion2 = nn.MSELoss()
optimizer = optim.SGD(net.parameters(), lr=self.dl_studio.
   ↪ learning_rate, momentum=self.dl_studio.momentum)

# Start training
print("\n\nStarting␣training␣loop...\n\n")
start_time = time.perf_counter()
labeling_loss_tally = []
regression_loss_tally = []
elapsed_time = 0.0
for epoch in range(self.dl_studio.epochs):
    print("")
    running_loss_labeling = 0.0
    running_loss_regression = 0.0
    for i, data in enumerate(self.train_dataloader):
        inputs, bbox_gt, labels = data[’image’], data[’bbox’],
           ↪ data[’label’]
        if i % 500 == 499:
            current_time = time.perf_counter()
            elapsed_time = current_time - start_time
            print("\n\n\n[epoch:%d/%d␣␣iter=%4d␣␣elapsed_time=%5d␣
               ↪ secs]␣␣␣␣␣␣Ground␣Truth:␣␣␣␣␣" %
                     (epoch+1, self.dl_studio.epochs, i+1,
                        ↪ elapsed_time)
                   + ’␣’.join(’%10s’ % self.dataserver_train.
                      ↪ class_labels[labels[j].item()]
                                                    for j in range(
                                                       ↪ self.
                                                       ↪ dl_studio.
                                                       ↪ batch_size))
                                                       ↪ )
        inputs = inputs.to(self.dl_studio.device)
        labels = labels.to(self.dl_studio.device)
        bbox_gt = bbox_gt.to(self.dl_studio.device)
        optimizer.zero_grad()
        if self.debug:
            self.dl_studio.display_tensor_as_image(
              torchvision.utils.make_grid(inputs.cpu(), nrow=4,
                 ↪ normalize=True, padding=2, pad_value=10))
        outputs = net(inputs)
        outputs_label = outputs[0]
        bbox_pred = outputs[1]
```

```
if i % 500 == 499:
    inputs_copy = inputs.detach().clone()
    inputs_copy = inputs_copy.cpu()
    bbox_pc = bbox_pred.detach().clone()
    bbox_pc_copy = bbox_pred.detach().clone()
    bbox_pc[bbox_pc<0] = 0
    bbox_pc[bbox_pc>1] = 1
    bbox_pc[torch.isnan(bbox_pc)] = 0
    _, predicted = torch.max(outputs_label.data, 1)
    print("[epoch:%d/%d␣␣iter=%4d␣␣elapsed_time=%5d␣secs]␣
        ↪ ␣Predicted␣Labels:␣␣␣␣␣" % (epoch+1, self.
        ↪ dl_studio.epochs, i+1, elapsed_time) + '␣'.join(
        ↪ '%10s' % self.dataserver_train.class_labels[
        ↪ predicted[j].item()] for j in range(self.
        ↪ dl_studio.batch_size)))
    for idx in range(self.dl_studio.batch_size):
        bbox_gt_copy = bbox_gt.detach().clone()
        bbox_gt_copy = bbox_gt_copy.cpu()
        bbox_pc_copy = bbox_pc_copy.cpu()
        i1 = bbox_gt_copy[idx][1]
        i2 = bbox_gt_copy[idx][3]
        j1 = bbox_gt_copy[idx][0]
        j2 = bbox_gt_copy[idx][2]
        k1 = bbox_pc_copy[idx][1]
        k2 = bbox_pc_copy[idx][3]
        l1 = bbox_pc_copy[idx][0]
        l2 = bbox_pc_copy[idx][2]
        [j1,i1,j2,i2] = [int(x) for x in utils.unnormalize
            ↪ ([j1,i1,j2,i2])]
        [l1,k1,l2,k2] = [int(x) for x in utils.unnormalize
            ↪ ([l1,k1,l2,k2])]
        print("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣gt_bb:␣␣[%d,%d,%d,%d]"
            ↪ %(j1,i1,j2,i2))
        print("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣pred_bb:␣␣[%d,%d,%d,%d]"
            ↪ %(l1,k1,l2,k2))
        try:
            inputs_copy[idx,0,i1:i2,j1] = 255
            inputs_copy[idx,0,i1:i2,j2] = 255
            inputs_copy[idx,0,i1,j1:j2] = 255
            inputs_copy[idx,0,i2,j1:j2] = 255
            inputs_copy[idx,2,k1:k2,l1] = 255
            inputs_copy[idx,2,k1:k2,l2] = 255
            inputs_copy[idx,2,k1,l1:l2] = 255
            inputs_copy[idx,2,k2,l1:l2] = 255
        except:
            print("index␣out␣of␣bound,␣Skipping")
loss_labeling = criterion1(outputs_label, labels)
loss_labeling.backward(retain_graph=True)
loss_regression = criterion2(bbox_pred, bbox_gt)
loss_regression.backward()
optimizer.step()
running_loss_labeling += loss_labeling.item()
running_loss_regression += loss_regression.item()
if i % 500 == 499:
    avg_loss_labeling = running_loss_labeling / float(500)
    avg_loss_regression = running_loss_regression / float
       ↪ (500)
    labeling_loss_tally.append(avg_loss_labeling)
    regression_loss_tally.append(avg_loss_regression)
    print("[epoch:%d/%d␣␣iter=%4d␣␣elapsed_time=%5d␣secs]␣
       ↪ ␣␣␣␣␣␣loss_labelling␣%.3f␣␣␣␣␣␣␣␣loss_regression
       ↪ :␣%.3f␣" % (epoch+1, self.dl_studio.epochs, i+1,
       ↪  elapsed_time, avg_loss_labeling,
       ↪ avg_loss_regression))
    FILE1.write("%.3f\n" % avg_loss_labeling)
    FILE1.flush()
    FILE2.write("%.3f\n" % avg_loss_regression)
    FILE2.flush()
    running_loss_labeling = 0.0
    running_loss_regression = 0.0
if i%500==499 and epoch == self.dl_studio.epochs-1:
    logger = logging.getLogger()
    old_level = logger.level
    logger.setLevel(100)
    plt.figure(figsize=[8,3])
    plt.imshow(np.transpose(torchvision.utils.make_grid(
       ↪ inputs_copy, normalize=False,
                                               padding=3,
                                                   ↪
                                                   ↪ pad_value
                                                   ↪ =255)
                                                   ↪ .cpu
                                                   ↪ (),
                                                   ↪ (1,2,0)
```

```
                                                                          ↪ ))
                        plt.show()
                        logger.setLevel(old_level)
            print("\nFinished␣Training\n")
            self.save_model(net)
            plt.figure(figsize=(10,5))
            plt.title("Labeling␣Loss␣vs.␣Iterations")
            plt.plot(labeling_loss_tally)
            plt.xlabel("iterations")
            plt.ylabel("labeling␣loss")
            plt.legend()
            plt.savefig("labeling_loss.png")
            plt.show()
            plt.title("regression␣Loss␣vs.␣Iterations")
            plt.plot(regression_loss_tally)
            plt.xlabel("iterations")
            plt.ylabel("regression␣loss")
            plt.legend()
            plt.savefig("regression_loss.png")
            plt.show()
```

```
invincible = tool(
    dataroot='/home/varun/work/courses/why2learn/hw/hw5/data',
    image_size=[64, 64],
    path_saved_model="/home/varun/work/courses/why2learn/hw/hw5/saves/
        ↪ saved_model.pt",
    momentum=0.9,
    learning_rate=1e-4,
    epochs=10,
    batch_size=4,
    classes=['cat','train','airplane'],
    use_gpu=True,
)


detector = tool.fearInoculum(dl_studio=invincible)
transform = tvt.Compose([tvt.ToTensor(),tvt.Normalize((0.5, 0.5, 0.5), (0.5,
    ↪  0.5, 0.5))])
```

```
# prepare train dataloader
coco = COCO('/home/varun/work/courses/why2learn/hw/annotations/
   ↪ instances_train2017.json')
dataserver_train = CocoDetection(transform, invincible.dataroot, invincible.
   ↪ class_labels, invincible.image_size, coco, loadDict=True, saveDict=
   ↪ False, mode="train")
detector.dataserver_train = dataserver_train
train_dataloader = torch.utils.data.DataLoader(dataserver_train, batch_size=
   ↪ invincible.batch_size, shuffle=True, num_workers=16)
detector.train_dataloader =train_dataloader

# model = detector.LOADnet2(skip_connections=True, depth=8)
model = pneumaNet(depth=16)

number_of_learnable_params = sum(
    p.numel() for p in model.parameters() if p.requires_grad
)
print(
    "\n\nThe␣number␣of␣learnable␣parameters␣in␣the␣model:␣%d"
    % number_of_learnable_params
)

num_layers = len(list(model.parameters()))
print("\nThe␣number␣of␣layers␣in␣the␣model:␣%d\n\n" % num_layers)


detector.run_code_for_training_with_CrossEntropy_and_MSE_Losses(model)
```

CODE-hw05_validation.py

```
import copy, time
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as tvt
import torch.optim as optim
from pycocotools.coco import COCO
import os, sys, logging
import matplotlib.pyplot as plt
# USER imports
```

```
sys.path.append("/home/varun/work/courses/why2learn/hw/DLStudio-2.1.6/")
from DLStudio import *

sys.path.append("/home/varun/work/courses/why2learn/hw/DLStudio-2.1.6/
   ↪ Examples")
from model import pneumaNet

from dataloader import CocoDetection
import utils

class tool(DLStudio):
    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)

    class fearInoculum(DLStudio.DetectAndLocalize):
        def __init__(self, dl_studio, dataserver_train=None,dataserver_test=
           ↪ None,dataset_file_train=None,dataset_file_test=None,):
            super().__init__(dl_studio, dataserver_train,dataserver_test,
               ↪ dataset_file_train,dataset_file_test,)
            self.dl_studio = dl_studio

        def run_code_for_testing_detection_and_localization(self, net):
            net.load_state_dict(torch.load(self.dl_studio.path_saved_model))
            correct = 0
            total = 0
            confusion_matrix = torch.zeros(len(self.dataserver_test.
               ↪ class_labels),
                                           len(self.dataserver_test.class_labels
                                              ↪ ))
            class_correct = [0] * len(self.dataserver_test.class_labels)
            class_total = [0] * len(self.dataserver_test.class_labels)
            with torch.no_grad():
                for i, data in enumerate(self.test_dataloader):
                    images, bounding_box, labels = data['image'], data['bbox'
                       ↪ ], data['label']
                    if len(labels) != 4:
                        continue
                    labels = labels.tolist()
                    if self.dl_studio.debug_test and i % 50 == 0:
                        print("\n\n[i=%d:]␣Ground␣Truth:␣␣␣␣␣" %i + '␣'.join('
                           ↪ %10s' %
                         self.dataserver_test.class_labels[labels[j]] for j in
                            ↪  range(self.dl_studio.batch_size)))
                    outputs = net(images)
                    outputs_label = outputs[0]
                    outputs_regression = outputs[1]
                    outputs_regression[outputs_regression < 0] = 0
                    outputs_regression[outputs_regression > 31] = 31
                    outputs_regression[torch.isnan(outputs_regression)] = 0
                    output_bb = outputs_regression.tolist()
                    _, predicted = torch.max(outputs_label.data, 1)
                    predicted = predicted.tolist()
                    if self.dl_studio.debug_test and i % 50 == 0:
                        print("[i=%d:]␣Predicted␣Labels:␣" %i + '␣'.join('%10s
                           ↪ ' %
                             self.dataserver_test.class_labels[predicted[j]]
                                ↪ for j in range(self.dl_studio.batch_size))
                                ↪ )
                        for idx in range(self.dl_studio.batch_size):
                            i1 = int(bounding_box[idx][1])
                            i2 = int(bounding_box[idx][3])
                            j1 = int(bounding_box[idx][0])
                            j2 = int(bounding_box[idx][2])
                            k1 = int(output_bb[idx][1])
                            k2 = int(output_bb[idx][3])
                            l1 = int(output_bb[idx][0])
                            l2 = int(output_bb[idx][2])
                            print("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣gt_bb:␣␣[%d,%d,%d,%d]"
                               ↪ %(j1,i1,j2,i2))
                            print("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣pred_bb:␣␣[%d,%d,%d,%d]"
                               ↪ %(l1,k1,l2,k2))
                            images[idx,0,i1:i2,j1] = 255
                            images[idx,0,i1:i2,j2] = 255
                            images[idx,0,i1,j1:j2] = 255
                            images[idx,0,i2,j1:j2] = 255
                            images[idx,2,k1:k2,l1] = 255
                            images[idx,2,k1:k2,l2] = 255
                            images[idx,2,k1,l1:l2] = 255
                            images[idx,2,k2,l1:l2] = 255
                        logger = logging.getLogger()
                        old_level = logger.level
                        logger.setLevel(100)
                        plt.figure(figsize=[8,3])
                        plt.imshow(np.transpose(torchvision.utils.make_grid(
                            ↪ images, normalize=True,
                                                                      padding=3,
                                                                         ↪
                                                                         ↪ pad_value
                                                                         ↪ =255)
                                                                         ↪ .cpu
                                                                         ↪ (),
                                                                         ↪ (1,2,0)
                                                                         ↪ ))
                        plt.show()
                        logger.setLevel(old_level)
                    for label,prediction in zip(labels,predicted):
                        confusion_matrix[label][prediction] += 1
                    total += len(labels)
                    correct += [predicted[ele] == labels[ele] for ele in range
                        ↪ (len(predicted))].count(True)
                    comp = [predicted[ele] == labels[ele] for ele in range(len
                        ↪ (predicted))]
                    for j in range(self.dl_studio.batch_size):
                        label = labels[j]
                        class_correct[label] += comp[j]
                        class_total[label] += 1
            print("\n")
            for j in range(len(self.dataserver_test.class_labels)):
                print('Prediction␣accuracy␣for␣%5s␣:␣%2d␣%%' % (
              self.dataserver_test.class_labels[j], 100 * class_correct[j] /
                 ↪ class_total[j]))
            print("\n\n\nOverall␣accuracy␣of␣the␣network␣on␣the␣1000␣test␣
               ↪ images:␣%d␣%%" %
                                                               (100 * correct
                                                                  ↪ / float(
                                                                  ↪ total)))
            print("\n\nDisplaying␣the␣confusion␣matrix:\n")
            out_str = "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"
            for j in range(len(self.dataserver_test.class_labels)):
                               out_str += "%15s" % self.dataserver_test.
                                   ↪ class_labels[j]
            print(out_str + "\n")
            for i,label in enumerate(self.dataserver_test.class_labels):
                out_percents = [100 * confusion_matrix[i,j] / float(
                    ↪ class_total[i])
                               for j in range(len(self.dataserver_test.
                                      ↪ class_labels))]
                        out_percents = ["%.2f" % item.item() for item in out_percents
                           ↪ ]
                        out_str = "%12s:␣␣" % self.dataserver_test.class_labels[i]
                        for j in range(len(self.dataserver_test.class_labels)):
                                                    out_str += "%15s" %
                                                       ↪ out_percents[j]
                        print(out_str)




invincible = tool(
    dataroot='/home/varun/work/courses/why2learn/hw/hw5/data',
    image_size=[64, 64],
    path_saved_model="/home/varun/work/courses/why2learn/hw/hw5/saves/
       ↪ saved_model.pt",
    momentum=0.9,
    learning_rate=1e-4,
    epochs=10,
    batch_size=4,
    classes=['cat','train','airplane'],
    use_gpu=True,
)


detector = tool.fearInoculum(dl_studio=invincible)

transform = tvt.Compose([tvt.ToTensor(),tvt.Normalize((0.5, 0.5, 0.5), (0.5,
   ↪  0.5, 0.5))])

# prepare test dataloader
coco = COCO('/home/varun/work/courses/why2learn/hw/annotations/
   ↪ instances_val2017.json')
dataserver_test = CocoDetection(transform, invincible.dataroot, invincible.
   ↪ class_labels, invincible.image_size, coco, loadDict=True, saveDict=
   ↪ False, mode="test")
detector.dataserver_test = dataserver_test
test_dataloader = torch.utils.data.DataLoader(dataserver_test, batch_size=
   ↪ invincible.batch_size, shuffle=False, num_workers=16)
detector.test_dataloader = test_dataloader
```

```
model = pneumaNet(depth=16)

detector.run_code_for_testing_detection_and_localization(model)
```

CODE-dataloader.py

```
from pycocotools.coco import COCO
import os
import numpy as np
import skimage.io as io
import matplotlib.pyplot as plt
import cv2

import torch
from torch.utils.data import Dataset
from PIL import Image
import os, glob, sys
import numpy as np
import utils, pickle

class CocoDetection(Dataset):
    def __init__(self,transform,dataPath,classes,size,coco,loadDict=False,
       ↪ saveDict=False,mode="test"):
        self.dataPath = dataPath
        self.class_labels = classes
        self.transform = transform

        # load pre-existing dictionary
        if loadDict and mode=="train":
            with open(os.path.join(dataPath,'dictTrain.pkl'), 'rb') as file:
                self.imgDict = pickle.load(file)
        elif loadDict and mode=="test":
            with open(os.path.join(dataPath,'dictTest.pkl'), 'rb') as file:
                self.imgDict = pickle.load(file)
        elif mode=="train":
            self.imgDict = utils.downloadCOCO(dataPath,classes,size,coco,
               ↪ saveDict)
        elif mode=="test":
            self.imgDict = utils.downloadCOCO(dataPath,classes,size,coco,
               ↪ saveDict)
        else:
            print("Something␣is␣wrong␣here␣!!!")
            sys.exit()
        self.imgDict = list(self.imgDict.values())

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

    def __getitem__(self,idx):
        img = self.imgDict[idx]
        label = torch.tensor(img[’classID’], dtype=torch.long)
        bbox = torch.tensor(img[’bbox’], dtype=torch.float)
        if self.transform:
            image = self.transform(img[’imgActual’])

        return {’image’:image, ’label’:label, ’bbox’:bbox}
```

CODE-model.py

```
import torch.nn as nn
import torch.nn.functional as F
import torch
import torch.nn as nn
import sys

sys.path.append("/home/varun/work/courses/why2learn/hw/DLStudio-2.1.6/")
from DLStudio import *


class SkipBlock(nn.Module):
    """
    Implementation of SkipBlock
    Inspired from Inception and resnet
    """

    def __init__(self, in_ch, out_ch):
        super().__init__()
        # self.downsample = downsample
        # self.skip_connections = skip_connections
        self.in_ch = in_ch
        self.out_ch = out_ch
        self.convo1x1 = nn.Conv2d(in_ch, out_ch, 1, stride=1, padding=1)
        self.convo3x3 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
        self.maxPool3x3 = nn.MaxPool2d(3, 1)
        self.bn = nn.BatchNorm2d(out_ch)
        self.reLU = nn.functional.relu

    def forward(self, x):
        identity = x
        # first
        out1 = self.convo3x3(x)
        out1 = self.convo3x3(out1)
        out1 = self.convo3x3(out1)
        # second
        out2 = self.maxPool3x3(x)
        out2 = self.convo1x1(out2)
        # add first and second
        out = out1 + out2
        out = self.bn(out)
        out = self.reLU(out)
        # skip connection
        out += identity
        return out


class pneumaNet(nn.Module):
    def __init__(self, depth=8):
        super().__init__()
        self.depth = depth
        # for classification
        self.sB128x128 = SkipBlock(128, 128)
        self.sB64x64 = SkipBlock(64, 64)

        self.convInx128 = nn.Conv2d(3, 128, 3, padding=1)
        self.conv128x64 = nn.Conv2d(128, 64, 3, padding=1)

        self.pool2x2 = nn.MaxPool2d(2, 2)
        self.reLU = nn.functional.relu
        self.fc1 = nn.Linear(65536, 1000)
        self.fc2 = nn.Linear(1000, 5)

        # for regression
        self.conv_seqn = nn.Sequential(
            nn.Conv2d(in_channels=128, out_channels=64, kernel_size=3,
                ↪ padding=1),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding
                ↪ =1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding
                ↪ =1),
            nn.ReLU(inplace=True),
        )
        self.fc_seqn = nn.Sequential(
            nn.Linear(65536, 1024),
            nn.ReLU(inplace=True),
            nn.Linear(1024, 512),
            nn.ReLU(inplace=True),
            nn.Linear(
                512, 4
            ), ## output for the 4 coords (x_min,y_min,x_max,y_max) of BBox
        )

    def forward(self, x):
        # the neck
        x = self.pool2x2(self.reLU(self.convInx128(x)))
        x1 = x.clone()

        ## network
        # four blocks
        for i in range(self.depth // 2):
            x1 = self.sB128x128(x1)
        # downsample
        x1 = self.conv128x64(x1)
        # four more blocks
        for i in range(self.depth // 2):
            x1 = self.sB64x64(x1)
        # head
        x1 = x1.view(-1, 65536)
        x1 = self.reLU(self.fc1(x1))
        x1 = self.fc2(x1)

        ## The Bounding Box regression
        x2 = self.conv_seqn(x)
        # flatten
        x2 = x2.view(x.size(0), -1)
        x2 = self.fc_seqn(x2)
        return x1, x2
```

CODE-utils.py

```
from pycocotools.coco import COCO
import os, sys
import numpy as np
import skimage.io as io
import matplotlib.pyplot as plt
import cv2

from PIL import Image
import requests
from io import BytesIO
import pickle

def normalize(data,min=0.0,max=63.0):
    return list((np.array(data) - min)/(max-min))

def unnormalize(data,min=0.0,max=63.0):
    return list(np.array(data)*(max-min)+min)

# funciton for checking if img follows the spec
def checkImgAnn(path_cls, classes, cls, img, catId, anns, size):
    filePath = os.path.join(path_cls, img[’file_name’])
    if os.path.exists(filePath):
        imgActual = Image.open(filePath)
    else:
        response = requests.get(img[’coco_url’])
        imgActual = Image.open(BytesIO(response.content))
        imgActual = imgActual.convert(mode=’RGB’)
        imgActual.save(filePath)

    ## Sepcs for a valid image
    # Spec 1: make sure class of largest object is equal to catID
    boxAreaMax = 0
    catIDmax = -1
    boxMax = []
    for ann in anns:
        area = ann[’bbox’][2] * ann[’bbox’][3]
        if area > boxAreaMax and ann[’category_id’] in coco.getCatIds(catNms
            ↪ =classes):
            boxAreaMax = area
            catIDmax = ann['category_id']
            boxMax = ann['bbox']
    if catIDmax != catId[0]:
        return False

    # Spec 2: check if widht and height of largest object are at least 1/3
        ↪ of image width and height
    # and bbox size is less than the image size
    w,h = imgActual.size
    if boxMax[2] < w/3 or boxMax[3] < h/3 or boxMax[2] > w or boxMax[3] > h:
        return False

    wScale, hScale = size[0]/w, size[1]/h
    bbox = [wScale*(boxMax[0])-1,hScale*(boxMax[1])-1,wScale*(boxMax[0]+
        ↪ boxMax[2])-1,hScale*(boxMax[1]+boxMax[3])-1]
    # can improve the logic here
    for i,coor in enumerate(bbox):
        if coor > 63:
            bbox[i] = 63
        elif coor < 0:
            bbox[i] = 0

    ## For valid images, return image dictionary
    imgDict = {}
    imgDict['classID'] = classes.index(cls)
    imgDict['catID'] = catId
    imgDict['bbox'] = normalize(bbox)
    imgDict['imgActual'] = imgActual.resize(size)
    return {filePath:imgDict}


# coco - instance of COCO class -> coco = COCO(jsonPath)
def downloadCOCO(root_path, classes, size=(128,128), coco=None, saveDict=
    ↪ False, mode="test"):
    # create a dictionary of images which can be used for training/
        ↪ validation
    dictImgs = {}
    # check if image folder already exists if not then create one
    for cls in classes:
        path_cls = os.path.join(root_path, cls)
        if not os.path.exists(path_cls):
            os.makedirs(path_cls)
        # load images
        catId = coco.getCatIds(catNms=cls)
        imgIds = coco.getImgIds(catIds=catId)
        imgs = coco.loadImgs(imgIds)

        # Start Downloading
        print("Downloading␣%d␣images␣for␣Class␣%s"%(len(imgIds),cls))
        for img in imgs:
            # get all annotations for the img
            annIds = coco.getAnnIds(imgIds=img[’id’])
            anns = coco.loadAnns(annIds)

            # download only if annotation are to the spec
            imgDict = checkImgAnn(path_cls, classes, cls, img, catId, anns,
                ↪ size)

            if imgDict != False:
                dictImgs.update(imgDict)

    # save the dictionary for faster access
    if saveDict==True and mode=="train":
        with open (os.path.join(root_path, ’dictTrain.pkl’),’wb’) as file:
            pickle.dump(dictImgs, file)
    elif saveDict==True and mode=="test":
        with open (os.path.join(root_path, ’dictTest.pkl’),’wb’) as file:
            pickle.dump(dictImgs, file)
    return dictImgs
```

4 Lessons Learned

The programming homework was straightforward and covered the basics of CNN implementation for object detection. Although, the programming took a while to complete, I did not find any major issues with implementation of CNN. I learned pycocotools for the first time but I wonder if it would be useful in the future.

5 Suggested Enhancements

FAQ, as posted on Piazza can be included as part of the assignment.