Meher Nagpal
EE104
Christopher Pham

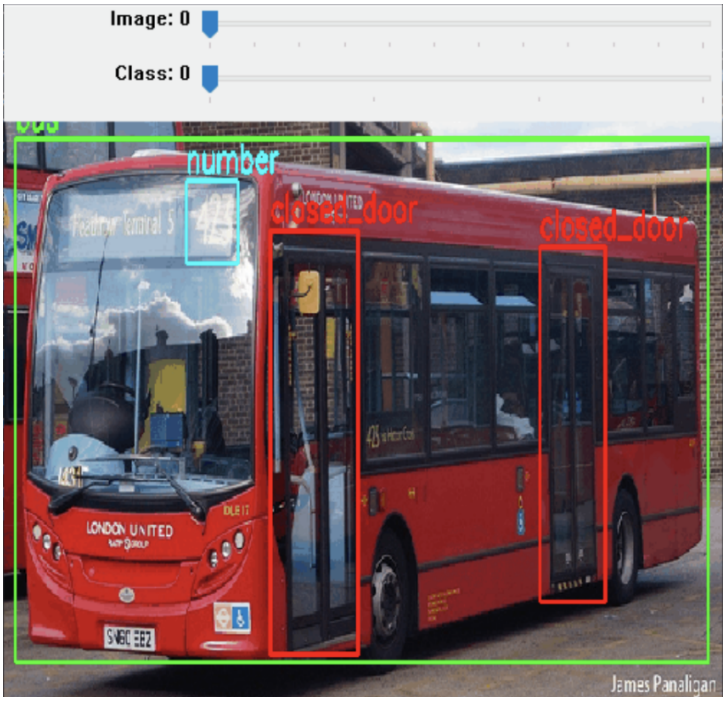
Lab Technical Document (Lab 7)

The purpose of this document is to demonstrate the code to other developers so that they can understand the code and develop it better to fix any bugs that may be introduced.

Code Highlights:

Pt 1 & 2: Train and recognize new object/myself:

This lab provides a comprehensive guide on training a custom object detection model using YOLOv5 with new images. The process is broken down into several key steps, which include installing a labeling tool, acquiring a YOLO COCO dataset, setting up YOLOv8 using Ultralytics, organizing the directory file structure, data cleaning, splitting the COCO dataset into train and validation sets, and finally, testing the new model.



First, the lab explains how to install a labeling tool, Modified Open Labeling, from GitHub. This tool allows users to label their custom images with bounding boxes, generating corresponding labels in the YOLO format. Required libraries are installed from the requirements.txt file to ensure proper functioning of the tool.

Next, the user is instructed to download the YOLO COCO dataset, which contains 80 predefined classes. The custom objects to be detected will be added to this dataset. The Ultralytics

framework is then set up, which includes installing the necessary files and setting up the directory structure. We added a couple more classes, including the new objects and my name.

```
ee104TrainImagePath = '/Users/mehernagpal/ultralytics/ultralytics/datasets/datasets/coco128/images/ee
ee104TrainLabelPath = '/Users/mehernagpal/ultralytics/ultralytics/datasets/datasets/coco128/labels/ee
ee104ValImagePath = '/Users/mehernagpal/ultralytics/ultralytics/datasets/datasets/coco128/images/ee1
ee104ValLabelPath = '/Users/mehernagpal/ultralytics/ultralytics/datasets/datasets/coco128/labels/ee1

coco_images = '/Users/mehernagpal/ultralytics/ultralytics/datasets/datasets/coco128/images/train2017
coco_labels = '/Users/mehernagpal/ultralytics/ultralytics/datasets/datasets/coco128/labels/train2017

#setup ratio (val ratio = rest of the files in origin dir after splitting into train and test)
train_ratio = 0.8
val_ratio = 0.2


#total count of imgs
totalImgCount = len(os.listdir(coco_images))
print("Total number of COCO images are : ",totalImgCount)

#storing files to corresponding arrays
for (dirname, dirs, files) in os.walk(coco_images):
    for filename in files:
        if filename.endswith('.jpg'):
            imgs.append(filename)
for (dirname, dirs, files) in os.walk(coco_labels):
    for filename in files:
        if filename.endswith('.txt'):
            xmls.append(filename)


#counting range for cycles
countForTrain = int(len(imgs)*train_ratio)
countForVal = int(len(imgs)*val_ratio)
print("Number of training images are   : ",countForTrain)
print("Number of validation images are : ",countForVal)



#cycle for train dir
for x in range(countForTrain):

    fileJpg = choice(imgs) # get name of random image from origin dir
    fileXml = fileJpg[:-4] +'.txt' # get name of corresponding annotation file

    shutil.copy(os.path.join(coco_images, fileJpg), os.path.join(ee104TrainImagePath, fileJpg))
    shutil.copy(os.path.join(coco_labels, fileXml), os.path.join(ee104TrainLabelPath, fileXml))
```

Once the framework is in place, the lab outlines the steps for data cleaning and splitting the COCO dataset into train and validation sets. The user is guided through inspecting and removing any extra files to ensure smooth operation of the script provided for splitting the data.

Finally, we are directed to test the installation and detection capabilities of the YOLOv8 model using various sources, such as images, webcams, and videos. The lab serves as a detailed guide for users looking to create custom object detection models using YOLOv5 and YOLOv8 with their own images.

Pt 3: Balloon Game

```
import pgzrun
import pygame
import pgzero
import random
from pgzero.builtins import Actor
from random import randint
```

Import different libraries used for PyGames

```
WIDTH = 800
HEIGHT = 600
balloon = Actor("balloon")
balloon.pos = 400, 300
bird = Actor("bird-up")
bird.pos = randint(800, 1600), randint(10, 200)
bird2 = Actor("bird-up")
bird2.pos = randint(400, 1600), randint(10, 200)
house = Actor("house")
house.pos = randint(800, 1600), 460
tree = Actor("tree")
tree.pos = randint(800, 1600), 450
```

Setting up the dimensions of the screen and setting up the different actors and where they spawn on the screen

```
bird_up = True
up = False
game_over = False
score = 0
number_of_updates = 0
scores = []
```

Setting up different callbacks that we could later flip to be true or false to go to a new function. We are also setting score to be 0 initially.

```
def update_high_scores():
    pass
    global score, scores
    filename = r"/Users/mehernagpal/Desktop/EE104/Lab7_Nagpal_Meher/P3_GameDevelopmen
    scores = []
    with open(filename, "r") as file:
        line = file.readline()
        high_scores = line.split()
        for high_score in high_scores:
            if(score > int(high_score)):
                scores.append(str(score) + " ")
                score = int(high_score)
            else:
                scores.append(str(high_score) + " ")
    with open(filename, "w") as file:
        for high_score in scores:
            file.write(high_score)
def display_high_scores():
    pass
    screen.draw.text("HIGH SCORES", (350, 150), color="black")
    y = 175
    position = 1
    for high_score in scores:
        screen.draw.text(str(position) + ". " + high_score, (350, y), color="black")
        y += 25
        position += 1
```

Setting up the high score of file and then replacing the high score if it was greater than any previous high score. Also displaying the high score when Game Over occurs.

```
def draw():
    screen.blit("background", (0, 0))
    if not game_over:
        balloon.draw()
        bird.draw()
        bird2.draw()
        house.draw()
        tree.draw()
        screen.draw.text("Score: " + str(score), (700, 5), color="black")
    else:
        display_high_scores()
```

Drawing the background and the different actors onto the screen

```
def on_mouse_down():
    global up
    up = True
    balloon.y -= 50
def on_mouse_up():
    global up
    up = False
```

When you click the balloon, the balloon goes up, when you don't click the balloon, the balloon goes down.

```
def update():
    global game_over, score, number_of_updates
    if not game_over:
```

Defining the different global updates for update class

```
    if not up:
        balloon.y += 1
    if bird.x > 0:
        bird.x -= 8
        if number_of_updates == 9:
            flap()
            number_of_updates = 0
        else:
            number_of_updates += 1
    else:
        bird.x = randint(800, 1600)
        bird.y = randint(10, 200)
        score += 1
        number_of_updates = 0

    if bird2.x > 0:
        bird2.x -= 4
        if number_of_updates == 9:
            flap()
            number_of_updates = 0
        else:
            number_of_updates += 1
    else:
        bird2.x = randint(800, 1600)
        bird2.y = randint(10, 200)
        score += 1
        number_of_updates = 0
```

Defining the speed of the birds and the positioning in which they spawn and having the birds move across the screen and give a point when they exit the screen.

```
    if house.right > 400:
        house.x -= 4
    else:
        if house.right > 396:
            house.x -= 4
            score += 1
        else:
            house.x -= 4
            if house.right <= 0:
                house.x = randint(800, 1600)
```

Have the house move at a certain speed and give you a point when you pass over the house

```
if balloon.top < 0 or balloon.bottom > 560:
    game_over = True
    update_high_scores()
if (balloon.collidepoint(bird.x, bird.y) or
        balloon.collidepoint(bird2.x, bird2.y) or
        balloon.collidepoint(house.x, house.y) or
        balloon.collidepoint(tree.x, tree.y)):
    game_over = True
    update_high_scores()
```

If the balloon touches the bottom or collides with any of the actors, it is game over