San Jose State University

Department of Electrical Engineering

EE104, Spring 2023, Pham

Laboratory Assignment #7

# Objectives

This lab allows you to use Yolov8 to recognize images from different sources. You will build a custom model to recognize your own images from webcams, pictures or video files from the internet or local drive.

You will also select to build one game, either a Baloon Flight or Sleeping Dragons. The Balloon Flight will give you a chance to learn about the coordination and background and foreground settings when you design a game to make it easy or difficult for a player. The Sleeping Dragons game design gives you a chance to learn about scheduler that can cause surprises and thus increase the game playing experience.

# Grading

Refer to the section **Python Programming** for grading criteria.

# Bibliography

I would like to acknowledge the Python open-source community and respective suppliers for making the material available.

# Download, Installation, and Licensing

You should not do the following steps in a hurry. **Slow down!** Read and follow instructions carefully so that you will not miss any crucial step.

## Install the tool to label custom images

Reference:

You will follow all steps from this tutorial up to Creating your YAML file to demonstrate that you can add labels to your new images:

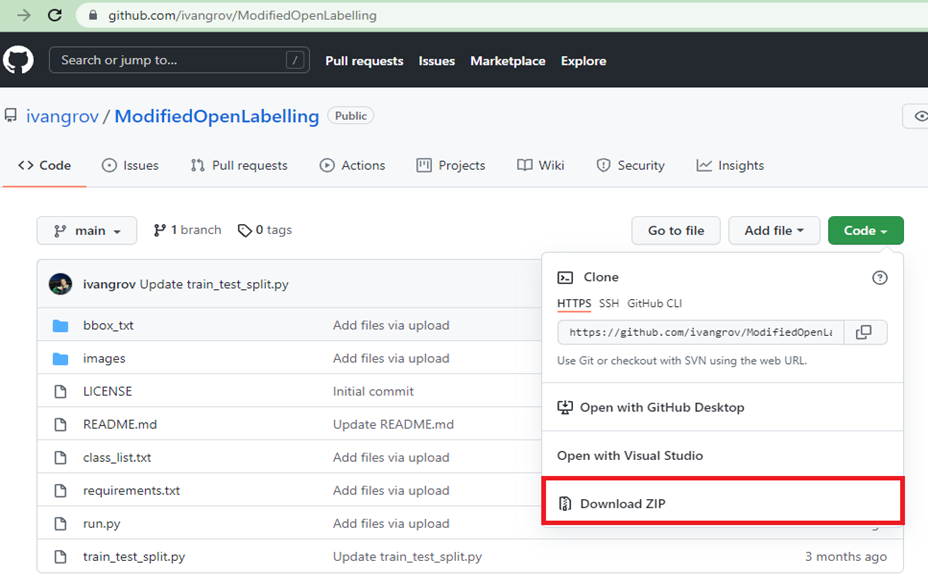
<https://wandb.ai/onlineinference/YOLO/reports/Collect-and-Label-Images-to-Train-a-YOLOv5-Object-Detection-Model-in-PyTorch--VmlldzoxMzQxODc3>

(i.e. you will not upload your files to wanb.ai website).

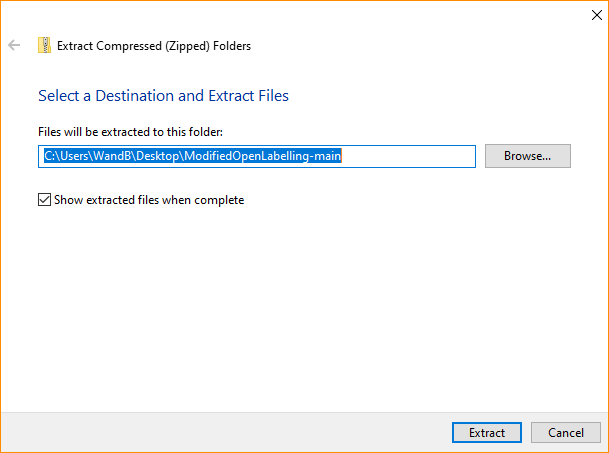
### Step 1: Get the Labeling Repo From GitHub

First you need to download and unzip the labeling repo from GitHub.

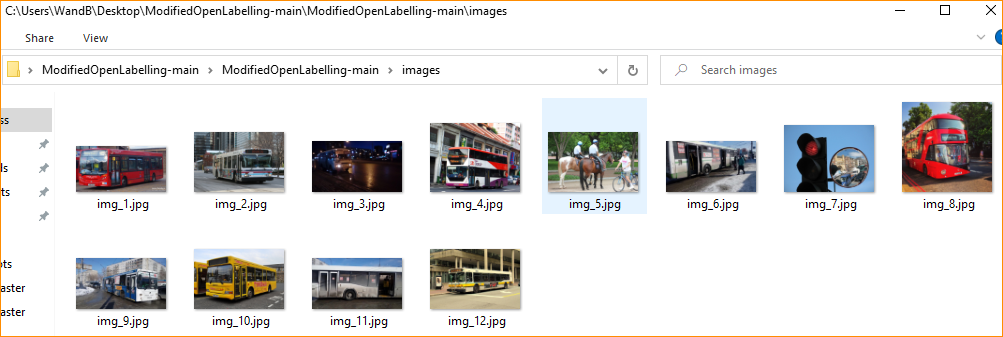
Here is the link<https://github.com/ivangrov/ModifiedOpenLabelling>



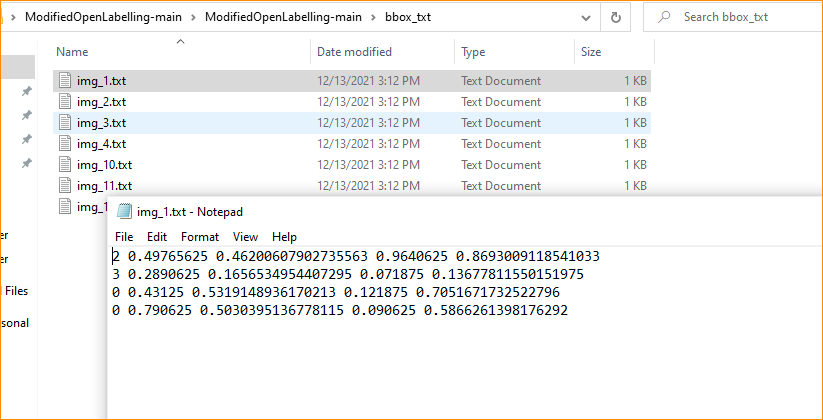
Once the download is complete, extract the folder.



You'll need to put the images to be labeled in the images folder.



The labels for each image in the YOLO format will be created with the same name but with a .txt extension in bbox\_txt folder.

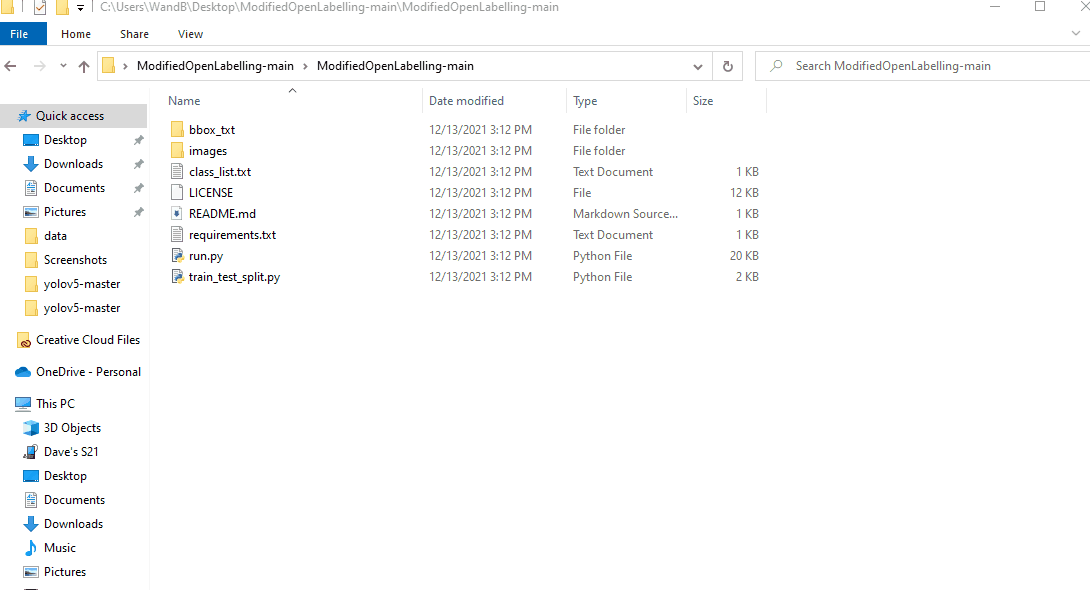


### Step 2: Install the required libraries listed inside the file requirements.txt file

Make sure that you have installed the required libraries in requirements.txt file to run the modified Open Labeling tool.

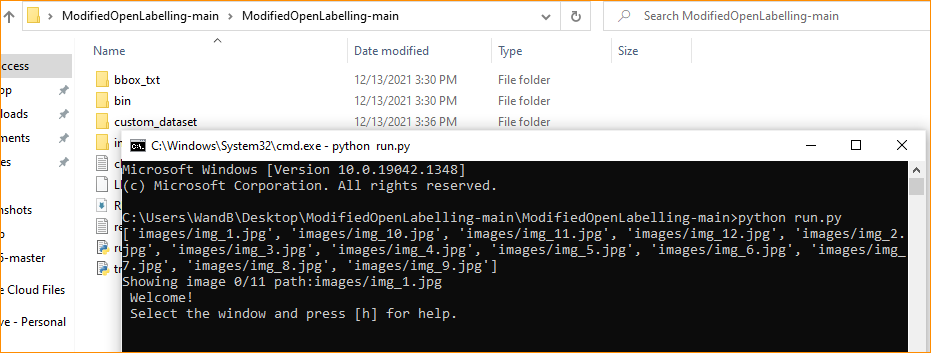
To do this simply open the folder location, enter **cmd** <enter> in the address bar and type:

**pip install -r requirements.txt**

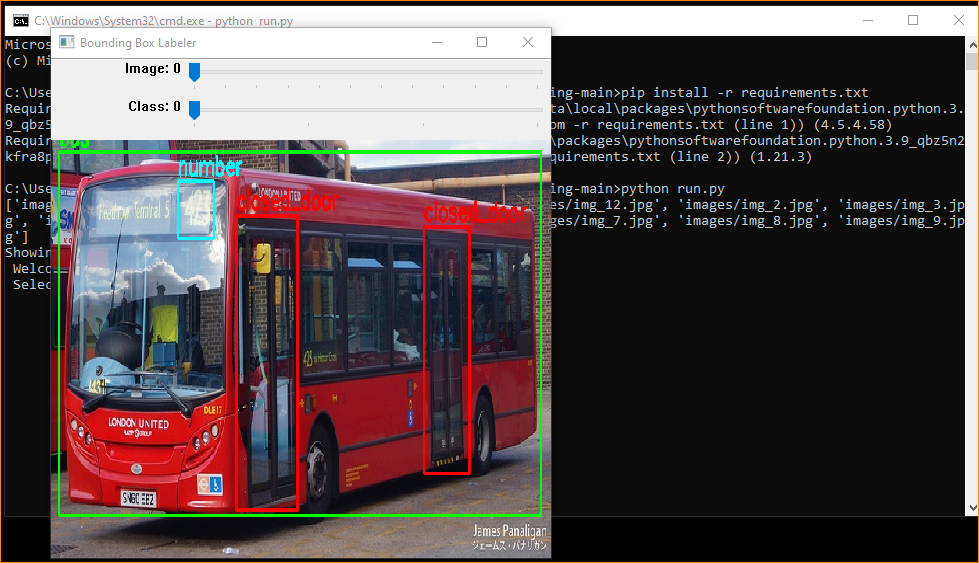


In order to launch the tool execute the run.py file enter:

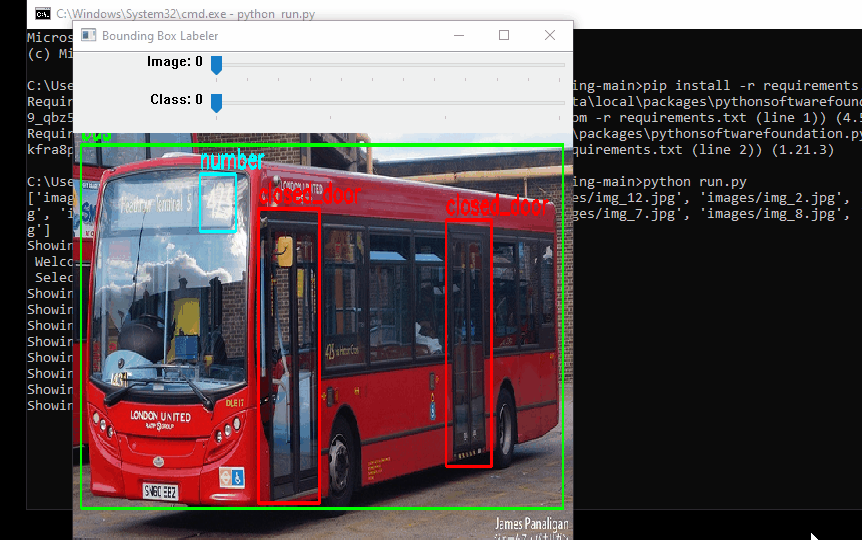
**python run.py**



Which should produce:



The sliding window bar at the top is used to switch the images.



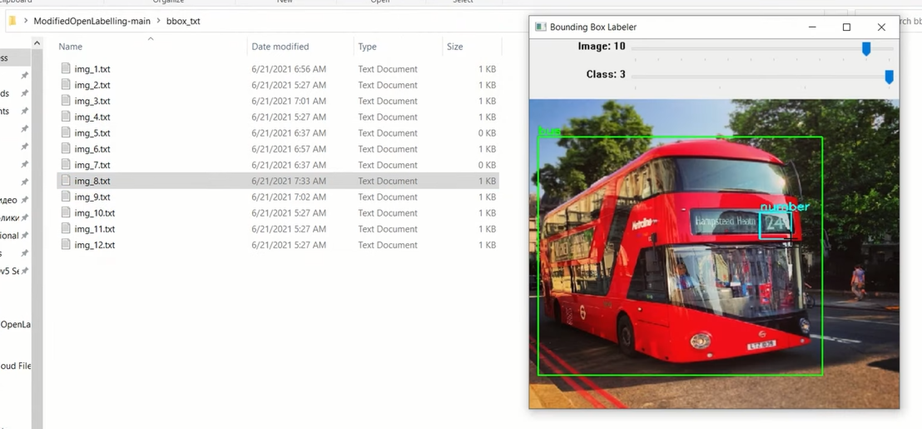
You can use these shortcut keys to navigate



The number of classes is specified in the class.txt file. In our case we have got 4 classes closed door, opened door, bus and number.



All the bounding box that you draw on images will be automatically added to .txt file



### Step 3: Delete any labeling mistakes

You can point to any label and right click to remove that label.

### Step 4: Stop the labeling process

You can just press Control-C from the Powershell window to stop the tool and the labeling process.

## Download a Yolo COCO dataset

Download and unzip the COCO (Common Object in COntext) dataset from here: <https://github.com/ultralytics/yolov5/releases/download/v1.0/coco128.zip>

The file is also available from Canvas in case the link above is down.

According to this website, you will see that this dataset has the following 80 classes indexed from 0 to 79. Your new objects will be added from 80, 81, 82, etc.

<https://github.com/ultralytics/yolov5/blob/d68afedb32fb5f3b632f67f2cbea2c89a145f0ad/data/coco128.yaml#L12-L13>

# class names

names: [ 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light',

'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',

'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',

'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard',

'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',

'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch',

'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',

'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear',

'hair drier', 'toothbrush', ‘your new objects will be added from here’ ]

## Download Yolov8 by installing Ultralytics, test run, add your own images, train your new model, and test again

* 1. Copy file yolov8n.pt and yolov8s.pt from Canvas to your local C:\Ultralytics directory. There are other yolov8x.pt files but they will require powerful CPU and computer resources to run these models. You can use them if your computer has high computing power and memories and other resources.
  2. Perform PART 1 from this file from Canvas: Yolov8\_Lab.txt

#######################################################################################

# PART 1: Install Ultralytics

#######################################################################################

#IMPORTANT: >>>>> Open Command Line or Powershell windows as an administrator<<<<<<<

**git clone https://github.com/ultralytics/ultralytics**

**pip install -e ultralytics**

**cd c:\ultralytics**

## Read and perform the steps in the [Directory File Structure](#_heading=h.gjdgxs) section below to understand the file structure.

Unzip the coco128.zip file and move the directory to under the dataset folder as shown in the picture below.

### Directory File Structure

Below is the directory file structure that the sample codes given in this lab is using. You can either create the same directory file structure or modify the code after your actual paths.

1. Note that there is a file **yolov5\_ee104\_split\_train\_val\_files.py** that you will run later to split the COCO dataset into 80/20 train/val ratio. Download this file from Canvas and save it to the C:\Ultralytics directory.
2. Unzip the COCO dataset that you downloaded earlier,

and move the folder coco128 into the subfolder datasets shown in the screenshot below.

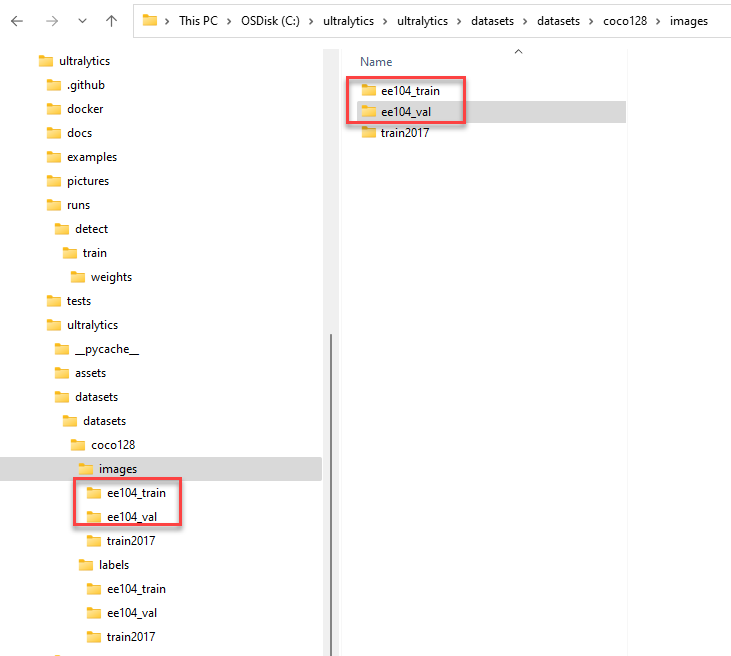
You will see the new directory train2017.

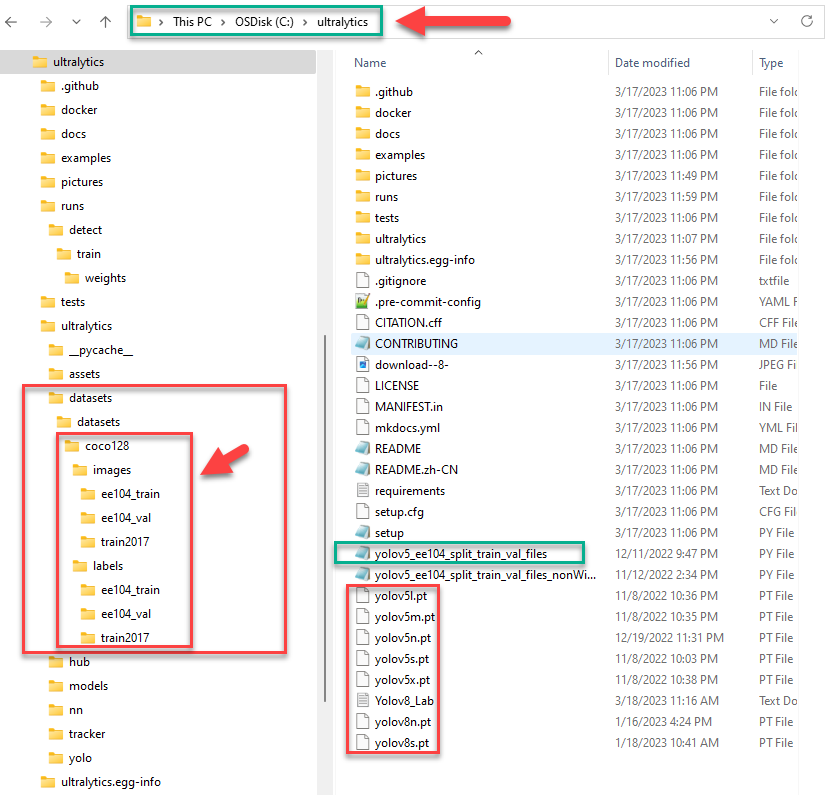
1. Create directories for your **ee104\_train** and **ee104\_val** files:

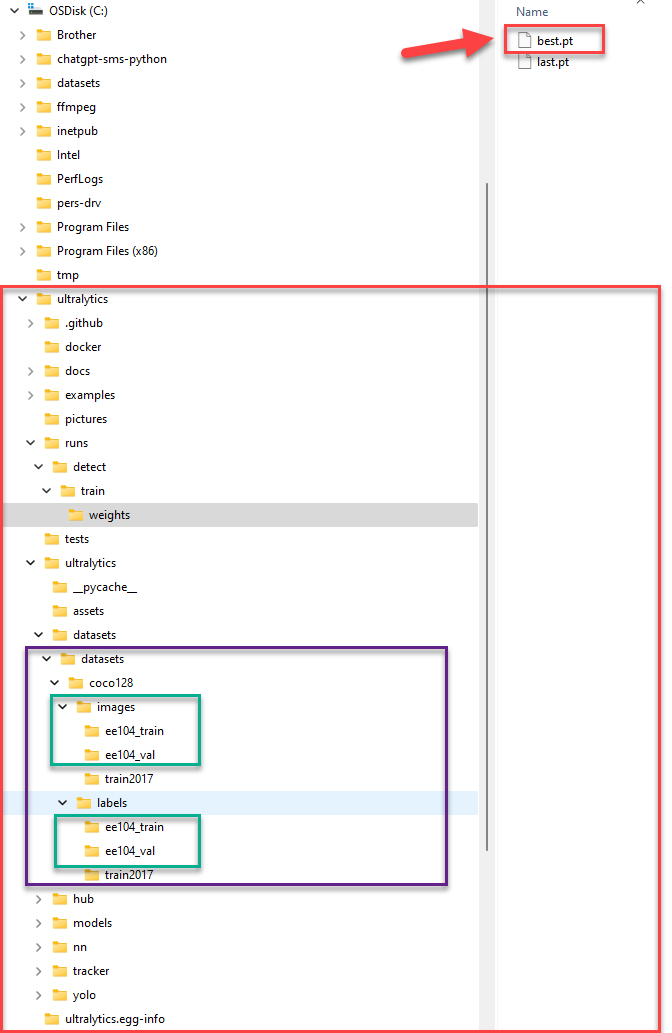
Now you must manually create the followings sub-directories if they are not already existing:

C:\ultralytics\ultralytics\datasets\datasets\coco128\images\ee104\_train

C:\ultralytics\ultralytics\datasets\datasets\coco128\images\ee104\_val







## Data Cleaning:

Now we will focus on these 2 directories:

C:\ultralytics\ultralytics\datasets\datasets\coco128\images\ee104\_train

C:\ultralytics\ultralytics\datasets\datasets\coco128\images\ee104\_val

1. Find the extra image files and extra txt files

Inspect two COCO source directories, you will see that

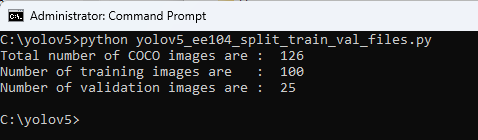
1. C:\ultralytics\ultralytics\datasets\datasets\coco128\images\train2017 has 2 images that do not have the same txt files in the C:\yolov5\datasets\coco128\labels\train2017 directory.
2. Also C:\ultralytics\ultralytics\datasets\datasets\coco128\labels\train2017 directory has 2 extra txt files with no equivalent images from the train2017 directory.
3. Clean those files: You will have to delete the extra files from both directories above. If not, the step below will not run correctly.

## Split the COCO source files into 80% Train & 20% Val directories

Note: The script below will not run correctly if you did not do the Data Cleaning step above.

1. Now you will run the provided file below to split the COCO source files into 80% for Train and 20% for Validation.

python yolov5\_ee104\_split\_train\_val\_files.py



1. Validate to see that the previously empty ee104\_train and ee140\_val directories are filled with images that you just copy over.

## Continue to perform the test run, add your own images, train your new model, and test again

Perform the rest from PART 1 from this file from Canvas: Yolov8\_Lab.txt

Note: copy and paste from the .txt file only. If you copy from the doc file, it will contain other hidden characters and the commands will not run correctly

**cd c:\ultralytics**

#################################################################################################

# PART 2: Test out that you installation is successful

#################################################################################################

# Reference: https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/

# Run the first detection using a picture on the web

**yolo task=detect mode=predict model=yolov8n.pt conf=0.25 source='https://cultivateconnections.org/wp-content/uploads/2020/07/Webp.net-resizeimage.jpg'**

# Detect a picture from your local hard drive (replace the path with your own path, use your own pictures)

**yolo task=detect mode=predict model=yolov8n.pt source="C:\ultralytics\pictures\dog\_cow.jpg"**

# Now detect from the computer USB webcam. Note that you have to go to the Windows Settings to activate the webcam

# you want to use if there is a built-in laptop webcam and one or more USB webcam(s).

**yolo task=detect mode=predict model=yolov8n.pt source=0 show=True**

Type Control-C from the Powershell window to stop the program.

# Now detect from an MP4 video

**yolo task=detect mode=predict model=yolov8n.pt source="https://dm0qx8t0i9gc9.cloudfront.net/watermarks/video/r6uQGb9/karate-school-with-trainers-and-young-boys-showing-fighting-techniques\_bl4bcqt7x\_\_ac3f8bcf740597f9b7e2757312aeac86\_\_P360.mp4" show=True**

Type Control-C from the Powershell window to stop the program.

# Note: The MP4 video can be on your local hard drive or an URL. Use Powershell for all commands below.

**yolo task=detect mode=predict model=yolov8n.pt source="C:\ultralytics\pictures\Visitors.mp4" show=True**

**yolo task=detect mode=predict model=yolov8n.pt source="https://youtu.be/aiubcUtd-h4" show=True**

#################################################################################################

# PART 3: Create the custom dataset

#################################################################################################

######################

# PART 3A: Edit the yaml file to add the custom class.

# Note in names index 80 below: train the model to recognize yourself

# In folder: C:\ultralytics\ultralytics\datasets\datasets, edit file coco128\_ee104.yaml

# ---------------------

# Below is the file coco128\_ee104.yaml

# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

# COCO128 dataset https://www.kaggle.com/ultralytics/coco128 (first 128 images from COCO train2017) by Ultralytics

# Example usage: python train.py --data coco128.yaml

# parent

# ├── yolov5

# └── datasets

# └── coco128 ← downloads here (7 MB)

# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]

path: C:/ultralytics/ultralytics/datasets/datasets/coco128 # dataset root dir

train: images/ee104\_train # train images (relative to 'path') 128 images

val: images/ee104\_val # val images (relative to 'path') 128 images

test: # test images (optional)

# Classes

names:

0: person

1: bicycle

2: car

3: motorcycle

4: airplane

5: bus

6: train

7: truck

8: boat

9: traffic light

10: fire hydrant

11: stop sign

12: parking meter

13: bench

14: bird

15: cat

16: dog

17: horse

18: sheep

19: cow

20: elephant

21: bear

22: zebra

23: giraffe

24: backpack

25: umbrella

26: handbag

27: tie

28: suitcase

29: frisbee

30: skis

31: snowboard

32: sports ball

33: kite

34: baseball bat

35: baseball glove

36: skateboard

37: surfboard

38: tennis racket

39: bottle

40: wine glass

41: cup

42: fork

43: knife

44: spoon

45: bowl

46: banana

47: apple

48: sandwich

49: orange

50: broccoli

51: carrot

52: hot dog

53: pizza

54: donut

55: cake

56: chair

57: couch

58: potted plant

59: bed

60: dining table

61: toilet

62: tv

63: laptop

64: mouse

65: remote

66: keyboard

67: cell phone

68: microwave

69: oven

70: toaster

71: sink

72: refrigerator

73: book

74: clock

75: vase

76: scissors

77: teddy bear

78: hair drier

79: toothbrush

80: Christopher Hoang Pham

# Notice that 80: Christopher Hoang Pham is my own class. You should have your own class(es) from 80, 81, 82, etc.

# Download script/URL (optional)

**download: https://ultralytics.com/assets/coco128.zip**

# ---------------------

# End of file coco128\_ee104.yaml

######################

# PART 3B: Add the images that you labeled.

# Train folder: C:\ultralytics\ultralytics\datasets\datasets\coco128\images\ee104\_train

# Validation folder: C:\ultralytics\ultralytics\datasets\datasets\coco128\images\ee104\_val

#

######################

# PART 3C: Train the model with your own images

# Open a Powershell window and run the commands below

cd c:\ultralytics

# Note: Do a trial run with epochs=20 first, then change the number of epochs

# to as high as your time permits, i.e. epochs=100

**yolo task=detect mode=train model=C:/ultralytics/ultralytics/models/v8/yolov8n.yaml data=C:/ultralytics/ultralytics/datasets/datasets/coco128\_ee104.yaml epochs=20**

# You will see the followings from the Window Powershell output.

# Note that last line of the output shows where the result is saved:

# Results saved to c:\ultralytics\runs\detect\train

# --------

from n params module arguments

0 -1 1 464 ultralytics.nn.modules.Conv [3, 16, 3, 2]

1 -1 1 4672 ultralytics.nn.modules.Conv [16, 32, 3, 2]

2 -1 1 7360 ultralytics.nn.modules.C2f [32, 32, 1, True]

3 -1 1 18560 ultralytics.nn.modules.Conv [32, 64, 3, 2]

4 -1 2 49664 ultralytics.nn.modules.C2f [64, 64, 2, True]

5 -1 1 73984 ultralytics.nn.modules.Conv [64, 128, 3, 2]

6 -1 2 197632 ultralytics.nn.modules.C2f [128, 128, 2, True]

7 -1 1 295424 ultralytics.nn.modules.Conv [128, 256, 3, 2]

8 -1 1 460288 ultralytics.nn.modules.C2f [256, 256, 1, True]

9 -1 1 164608 ultralytics.nn.modules.SPPF [256, 256, 5]

10 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']

11 [-1, 6] 1 0 ultralytics.nn.modules.Concat [1]

12 -1 1 148224 ultralytics.nn.modules.C2f [384, 128, 1]

13 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']

14 [-1, 4] 1 0 ultralytics.nn.modules.Concat [1]

15 -1 1 37248 ultralytics.nn.modules.C2f [192, 64, 1]

16 -1 1 36992 ultralytics.nn.modules.Conv [64, 64, 3, 2]

17 [-1, 12] 1 0 ultralytics.nn.modules.Concat [1]

18 -1 1 123648 ultralytics.nn.modules.C2f [192, 128, 1]

19 -1 1 147712 ultralytics.nn.modules.Conv [128, 128, 3, 2]

20 [-1, 9] 1 0 ultralytics.nn.modules.Concat [1]

21 -1 1 493056 ultralytics.nn.modules.C2f [384, 256, 1]

22 [15, 18, 21] 1 897664 ultralytics.nn.modules.Detect [80, [64, 128, 256]]

YOLOv8n summary: 225 layers, 3157200 parameters, 3157184 gradients, 8.9 GFLOPs

Ultralytics YOLOv8.0.54 Python-3.10.8 torch-1.13.0+cpu CPU

yolo\engine\trainer: task=detect, mode=train, model=C:/ultralytics/ultralytics/models/v8/yolov8n.yaml, data=C:/ultralytics/ultralytics/datasets/datasets/coco128\_ee104.yaml, epochs=100, patience=50, batch=16, imgsz=640, save=True, save\_period=-1, cache=False, device=None, workers=8, project=None, name=None, exist\_ok=False, pretrained=False, optimizer=SGD, verbose=True, seed=0, deterministic=True, single\_cls=False, image\_weights=False, rect=False, cos\_lr=False, close\_mosaic=10, resume=False, overlap\_mask=True, mask\_ratio=4, dropout=0.0, val=True, split=val, save\_json=False, save\_hybrid=False, conf=None, iou=0.7, max\_det=300, half=False, dnn=False, plots=True, source=None, show=False, save\_txt=False, save\_conf=False, save\_crop=False, hide\_labels=False, hide\_conf=False, vid\_stride=1, line\_thickness=3, visualize=False, augment=False, agnostic\_nms=False, classes=None, retina\_masks=False, boxes=True, format=torchscript, keras=False, optimize=False, int8=False, dynamic=False, simplify=False, opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01, momentum=0.937, weight\_decay=0.0005, warmup\_epochs=3.0, warmup\_momentum=0.8, warmup\_bias\_lr=0.1, box=7.5, cls=0.5, dfl=1.5, fl\_gamma=0.0, label\_smoothing=0.0, nbs=64, hsv\_h=0.015, hsv\_s=0.7, hsv\_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.0, copy\_paste=0.0, cfg=None, v5loader=False, tracker=botsort.yaml, save\_dir=c:\ultralytics\runs\detect\train

Overriding model.yaml nc=80 with nc=81

from n params module arguments

0 -1 1 464 ultralytics.nn.modules.Conv [3, 16, 3, 2]

1 -1 1 4672 ultralytics.nn.modules.Conv [16, 32, 3, 2]

2 -1 1 7360 ultralytics.nn.modules.C2f [32, 32, 1, True]

3 -1 1 18560 ultralytics.nn.modules.Conv [32, 64, 3, 2]

4 -1 2 49664 ultralytics.nn.modules.C2f [64, 64, 2, True]

5 -1 1 73984 ultralytics.nn.modules.Conv [64, 128, 3, 2]

6 -1 2 197632 ultralytics.nn.modules.C2f [128, 128, 2, True]

7 -1 1 295424 ultralytics.nn.modules.Conv [128, 256, 3, 2]

8 -1 1 460288 ultralytics.nn.modules.C2f [256, 256, 1, True]

9 -1 1 164608 ultralytics.nn.modules.SPPF [256, 256, 5]

10 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']

11 [-1, 6] 1 0 ultralytics.nn.modules.Concat [1]

12 -1 1 148224 ultralytics.nn.modules.C2f [384, 128, 1]

13 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']

14 [-1, 4] 1 0 ultralytics.nn.modules.Concat [1]

15 -1 1 37248 ultralytics.nn.modules.C2f [192, 64, 1]

16 -1 1 36992 ultralytics.nn.modules.Conv [64, 64, 3, 2]

17 [-1, 12] 1 0 ultralytics.nn.modules.Concat [1]

18 -1 1 123648 ultralytics.nn.modules.C2f [192, 128, 1]

19 -1 1 147712 ultralytics.nn.modules.Conv [128, 128, 3, 2]

20 [-1, 9] 1 0 ultralytics.nn.modules.Concat [1]

21 -1 1 493056 ultralytics.nn.modules.C2f [384, 256, 1]

22 [15, 18, 21] 1 906541 ultralytics.nn.modules.Detect [81, [64, 128, 256]]

YOLOv8n summary: 225 layers, 3166077 parameters, 3166061 gradients, 8.9 GFLOPs

**yolo task=detect mode=train model=C:/ultralytics/ultralytics/models/v8/yolov8n.yaml data=C:/ultralytics/ultralytics/datasets/datasets/coco128\_ee104.yaml epochs=100**

TensorBoard: Start with 'tensorboard --logdir c:\ultralytics\runs\detect\train', view at http://localhost:6006/

optimizer: SGD(lr=0.01) with parameter groups 57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias

train: Scanning C:\ultralytics\ultralytics\datasets\datasets\coco128\labels\ee104\_train... 144 images, 0 b

train: New cache created: C:\ultralytics\ultralytics\datasets\datasets\coco128\labels\ee104\_train.cache

val: Scanning C:\ultralytics\ultralytics\datasets\datasets\coco128\labels\ee104\_val... 36 images, 1 backgr

val: New cache created: C:\ultralytics\ultralytics\datasets\datasets\coco128\labels\ee104\_val.cache

Plotting labels to c:\ultralytics\runs\detect\train\labels.jpg...

Image sizes 640 train, 640 val

Using 0 dataloader workers

Logging results to c:\ultralytics\runs\detect\train

Starting training for 100 epochs...

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

1/100 0G 3.528 5.699 4.272 176 640: 100%|██████████| 9/9 [01:11

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0 0 0 0

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

2/100 0G 3.391 5.641 4.266 173 640: 100%|██████████| 9/9 [01:11

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0 0 0 0

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

3/100 0G 3.563 5.638 4.235 217 640: 100%|██████████| 9/9 [01:10

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0 0 0 0

...

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

92/100 0G 2.352 4.168 2.636 96 640: 100%|██████████| 9/9 [01:08

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0.517 0.0347 0.0534 0.0363

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

93/100 0G 2.335 4.194 2.645 85 640: 100%|██████████| 9/9 [01:08

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0.392 0.0638 0.0593 0.0388

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

94/100 0G 2.293 4.085 2.617 95 640: 100%|██████████| 9/9 [01:08

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0.397 0.0673 0.06 0.0384

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

95/100 0G 2.292 4.079 2.589 188 640: 100%|██████████| 9/9 [01:08

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0.477 0.0347 0.0601 0.0379

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

96/100 0G 2.3 4.081 2.598 101 640: 100%|██████████| 9/9 [01:09

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0.474 0.0347 0.0426 0.0292

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

97/100 0G 2.282 4.034 2.593 63 640: 100%|██████████| 9/9 [01:08

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0.475 0.0373 0.0522 0.0345

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

98/100 0G 2.223 3.966 2.559 59 640: 100%|██████████| 9/9 [01:08

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0.395 0.048 0.0595 0.0387

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

99/100 0G 2.279 3.98 2.558 87 640: 100%|██████████| 9/9 [01:08

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0.409 0.0773 0.06 0.0407

Epoch GPU\_mem box\_loss cls\_loss dfl\_loss Instances Size

100/100 0G 2.26 4.002 2.562 59 640: 100%|██████████| 9/9 [01:08

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0.409 0.0773 0.0804 0.0517

100 epochs completed in 2.193 hours.

Optimizer stripped from c:\ultralytics\runs\detect\train\weights\last.pt, 6.5MB

Optimizer stripped from c:\ultralytics\runs\detect\train\weights\best.pt, 6.5MB

Validating c:\ultralytics\runs\detect\train\weights\best.pt...

Ultralytics YOLOv8.0.54 Python-3.10.8 torch-1.13.0+cpu CPU

YOLOv8n summary (fused): 168 layers, 3160775 parameters, 0 gradients, 8.8 GFLOPs

Class Images Instances Box(P R mAP50 mAP50-95): 100%|██████████|

all 36 100 0.409 0.0773 0.0804 0.0517

person 36 28 0 0 0.0143 0.00371

car 36 3 1 0 0 0

airplane 36 1 1 0 0 0

dog 36 4 1 0 0 0

elephant 36 4 1 0 0 0

giraffe 36 2 0 0 0.0192 0.00696

tie 36 1 1 0 0 0

frisbee 36 1 0 0 0 0

sports ball 36 2 0 0 0 0

baseball bat 36 1 0 0 0 0

skateboard 36 4 0 0 0 0

tennis racket 36 1 0 0 0 0

cup 36 4 1 0 0 0

fork 36 1 0 0 0 0

bowl 36 1 0.392 1 0.995 0.572

carrot 36 2 1 0 0 0

donut 36 13 0 0 0 0

cake 36 1 1 0 0 0

toilet 36 1 0 0 0 0

tv 36 1 0 0 0 0

laptop 36 2 0 0 0 0

cell phone 36 1 0 0 0 0

teddy bear 36 3 1 0 0 0

toothbrush 36 3 0 0 0 0

Christopher Hoang Pham 36 15 0.836 0.933 0.982 0.709

Speed: 2.1ms preprocess, 133.5ms inference, 0.0ms loss, 6.8ms postprocess per image

Results saved to c:\ultralytics\runs\detect\train

# --------

##################################################################################################

# PART 4: Test out that you custom training is successful

##################################################################################################

# The purpose is to see that you can detect your own face now from the computer USB webcam or a picture with your face.

# Note that now you will use your own best.pt model that you just trained.

# This model is in the folder: C:\ultralytics\runs\detect\train\weights

# Now detect from the computer USB webcam

**yolo task=detect mode=predict model=C:/ultralytics/runs/detect/train/weights/best.pt source=0 show=True**

# Detect a picture from your local hard drive (replace the path with your own path, use YOUR\_OWN\_PICTURE)

**yolo task=detect mode=predict model=C:/ultralytics/runs/detect/train/weights/best.pt source="C:\ultralytics\pictures\YOUR\_OWN\_PICTURE.jpg"**

# For example, I am using my own picture from this folder C:\ultralytics\ultralytics\datasets\datasets\coco128\images\ee104\_train

**yolo task=detect mode=predict model=C:/ultralytics/runs/detect/train/weights/best.pt source="C:/ultralytics/ultralytics/datasets/datasets/coco128/images/ee104\_train\2.jpg"**

PS C:\ultralytics> **yolo task=detect mode=predict model=C:/ultralytics/runs/detect/train/weights/best.pt source="C:/ultralytics/ultralytics/datasets/datasets/coco128/images/ee104\_train\2.jpg"**

Ultralytics YOLOv8.0.54 Python-3.10.8 torch-1.13.0+cpu CPU

YOLOv8n summary (fused): 168 layers, 3160775 parameters, 0 gradients, 8.8 GFLOPs

image 1/1 C:\ultralytics\ultralytics\datasets\datasets\coco128\images\ee104\_train\2.jpg: 640x544 1 Christopher Hoang Pham, 115.2ms

Speed: 1.0ms preprocess, 115.2ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 640)

PS C:\ultralytics>

# >>>>>>>> Notice that it detects my face as "Christopher Hoang Pham" <<<<<<<<<

# Now detect from an MP4 video. Use YOUR\_OWN\_MP4\_VIDEO.mp4

**yolo task=detect mode=predict model=C:/ultralytics/runs/detect/train/weights/best.pt**

**source="C:/somefolder/YOUR\_OWN\_MP4\_VIDEO.mp4" show=True**

# Note: The MP4 video can be on your local hard drive or an URL. Use Powershell for all commands below.

**yolo task=detect mode=predict model=C:/ultralytics/runs/detect/train/weights/best.pt**

**source="https://www.somewhere.com/YOUR\_OWN\_MP4\_VIDEO.mp4" show=True**

##################################################################################################

# NOTES: Overtrained

##################################################################################################

# If you have too many of your own pictures compared to other pictures of other classes in your yaml file

# then chances you are overtraining your customized model, i.e. it will only detect your face and ignore the

# rest. That's OK for learning purpose because you are proving that you can detect your own picture now.

# For practical applications, you want to balance the number of pictures for each class so not to overtrain your model.

##################################################################################################

# DONE: You are now completed the training and testing of your own customized model using Yolov8.

#################################################################################################

# Your Deliverables

|  |  |  |
| --- | --- | --- |
| **Program or Requirement** | **Use Case** | **Earned Score / Max Score** |
| Demonstration Video | You must submit a demonstration video or your score for this lab will be zero. See below for more information about the demonstration video. |  |
| Document file & check in your README file, documentation, video and codes to Github | README is a brief user guide and developer documentation so that the user can install the proper python packages and knows how to execute your program.  The documentation section can contain sample screenshots with explanation. | \_\_\_\_\_ / 10 |
| Train and recognize a new object | Add and train successfully one more class of your choice to the 80 existing classes. Add a minimum 15-30 pictures for accuracy.  To increase the accuracy, you can add as many picture as you want.  **In the demonstration video, hold your object and demonstrate that the AI can recognize this new class.** | \_\_\_\_\_ / 30 |
| Train and recognize yourself | Add and train successfully one more class to the 81 existing classes to recognize yourself. Add a minimum 15-30 pictures for accuracy.  To increase the accuracy, you can add as many picture as you want.  **In the demonstration video, hold your object and demonstrate that the AI can recognize this new class and also yourself!** | \_\_\_\_\_ / 30 |
| Game Development – Balloon Flight or Sleeping Dragons | The base code is at the end of the book inside the Reference section.  Use cases for entertainment and educational applications: Leverage the base code and add three of your own Hacks and Tweaks to add the following features (10 points each). Note that you must demonstrate that you can play your own game here in case you need to help your sale department to create live demo!   * Balloon Flight: Play for at least 2 minutes and two of any of these choices: Lives, More High Scores, File Handling, Level Up, Space out the Obstacles, * Sleeping Dragons: Add Another Hero, Less Predictable Dragons, Play for at least 2 minutes | \_\_\_\_\_ / 30 |
|  | **TOTAL** | **100%** |

That’s all for this lab. Hopefully you found it useful and increase your interest in the Python world! See you in the next lab.

# Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. You are required to upload YouTube videos to demonstrate your working solutions. You are also required to submit an archive of your project in the form of a ZIP file. Use 7-Zip option to create the ZIP file. Name the archive lab#\_yourlastname\_yourfirstname.zip. Refer to Lab 1 for detailed instructions.

You will submit your zip file to the instructor through Canvas by the due date and time. If your program is not completely functional by the due date, you should demonstrate and turn in what you have accomplished to receive partial credit. See the syllabus for the late penalty guideline