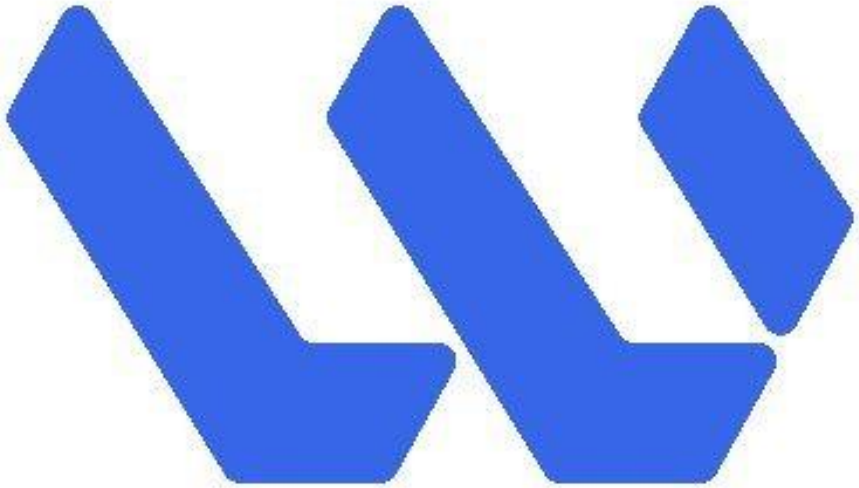
Wobot.ai

Hackathon Report Submission



Submitted By **:** Mansi Binjola

Date of Submission: 25th August, 2021

Stage 1 Evaluation

Task: Train the given hat_hardhat_dataset to any deep learning model and inference it with the given video.

My Approach:

I trained the following dataset with the yolov4 Object Detection Model.

Why yolov4?

YOLO is short for “You Only Look Once”. It is a real-time object recognition system that can recognize multiple objects in a single frame. YOLO is based on a single Convolutional Neural Network (CNN). The CNN divides an image into regions and then it predicts the boundary boxes and probabilities for each region.
YOLOv4 is an enhanced version of YOLOv3.
It is twice as fast as the EfficientDet (competitive recognition model) with comparable performance. In addition, AP (Average Precision) and FPS (Frames Per Second) increased by 10% and 12% compared to YOLOv3.

The steps to train a yolov4 are as follows:-

1. Data Preparation

- Downloading the dataset

Using chrome browser I download the train and test zip files and unzip in a folder.
Inside the HardHat_Dataset.zip consist of two folders:-

- Images
- annotations

- Converting the annotations to .csv file
  - The given annotations were inside xml files which were of pascal voc format. By using xml.etree.ElementTree package, I was able to parse the xml files and save it inside a DataFrame df.

- I created a DataFrame using the pandas package and extracted filename, width,height, class, xmin, ymin, xmax and ymax from the xml file.
- The function xmlParser() accepts 3 inputs and parses the xml file and converts it to DataFrame.

The code is given below

```
import xml.etree.ElementTree as ET
import pandas as pd
import os

def xmlParser(path,filename,df):
   '''
   parses the xml file and converts it to appropriate dataFrame format

   input:   path,filename,current dataFrame
   process: parses the xml file
   return:  DataFrame

   '''
   tree = ET.parse(path+filename)
   root = tree.getroot()
   fname = root.find("filename").text
   size = root.find("size")
   w,h = size[0].text,size[1].text
   for elem in root.findall("object"):
       name = elem.find("name").text
       bndbox = elem.find("bndbox")
       xmin,ymin,xmax,ymax = bndbox[0].text, bndbox[1].text,
bndbox[-2].text, bndbox[-1].text

       df =
df.append({"filename":fname,"width":w,"height":h,"class":name,"xmin":xmi
n,"ymin":ymin,"xmax":xmax,"ymax":ymax},
                       ignore_index=True)
   return(df)
# print(df.head())

cnt = 0
df =
```

```
pd.DataFrame(columns=("filename","width","height","class","xmin","ymin",
"xmax","ymax"))
path = "annotations/"
files = os.listdir(path)
for file in files:
   if(file.endswith(".xml")):
       df = xmlParser(path,file,df)
       cnt+=1

print(len(df),cnt)
```

- Removing data

Since we needed to make a head and helmet model, we needed to remove additional data from the dataFrame. Using the df.to_csv() function in pandas I converted the Data Frame into a csv format file.

```
df = df[df[class]!=”person”]]
df.to_csv("train.csv",index=False)
```

- Converting the .csv file to yolov4 format files

For making data ready for training, we need to convert the annotations into yolov4 format.

The yolov4 format is as follows:

`object-class` `x_center` `y_center` `width` `height`

where,

object-class: integer number for an object(from 0 to n)

x_center, **y_center**, **width**, **height**: float values relative to width and height of the image, will be between 0.0 to 1.0

The function Bndd2YoloLine() converts the annotations in yolov4 format and saves each image’s bounding boxes in the same name of the .txt file.

```
import pandas as pd
import os
```

```
def Bnddf2YoloLine(df,fname, classList=["head","helmet"]):
    '''
    input:df of particular image, fname= image name
    process: converts the .csv file data into yolo format data as saves
file in path+file_name.txt
    returns: none
    '''
    f = open("yoloformat/"+fname[:-3]+"txt", "a")
    for cnt in range(len(df)):
        xmin = df.iloc[cnt]['xmin']
        xmax = df.iloc[cnt]['xmax']
        ymin = df.iloc[cnt]['ymin']
        ymax = df.iloc[cnt]['ymax']

        xcen = float((xmin + xmax)) / 2 / df.iloc[cnt]['width']
        ycen = float((ymin + ymax)) / 2 / df.iloc[cnt]['height']

        w = float((xmax - xmin)) / df.iloc[cnt]['width']
        h = float((ymax - ymin)) / df.iloc[cnt]['height']


        dfName = df.iloc[cnt]['class']

        classIndex = classList.index(dfName)
        f.writelines(str(classIndex)+" "+str(xcen)+" "+str(ycen)+"
"+str(w)+" "+str(h)+"\n")
    f.close()



path = "jpg_images/"
#read csv file
df = pd.read_csv("head_helmet.csv")
# print(len(df))

cnt = 0

#for every image inside path make yolo format .txt files
for fname in os.listdir(path):
    Bnddf2YoloLine(df.loc[df['filename']==fname],fname)
    cnt+=1
```

```
print("number of files created: "+str(cnt))
```

- ## Converting dataset images to .JPG format

Since yolov4 can only train images in .jpg images, So I converted the images in .jpg format using the function convtPNGtoJPG(path,dest)

```
import os
from PIL import Image

def convtPNGtoJPG(path,dest):
   '''

   Function which converts all the png images present in the path to
.jpg format images

   input: path of images to be converted,dest = to save the converted
images in destination file
   process: converts .png image to .jpg image
   return: none

   '''

   cnt = 0
   for file in os.listdir(path):
       if(file.endswith(".png")):
           im = Image.open(path+file)
           rgb_im = im.convert('RGB')
           rgb_im.save(dest+file[:-3]+"jpg")
           #print(cnt)
           cnt+=1

convtPNGtoJPG("images/","images_in_jpg/")
```

- ## Creating necessary files for model training

For yolov4 model training we need these files. Without them no model can be trained.

  - a file which has class names, here head and helmet and saved as obj.names.

```
f = open("obj.names","a")
f.write("head\nhelmet")
f.close()
```

- Obj.data file: consists of details of the model which is required during training.
- Create a backup folder(here helmet_training) for saving weights. If the model is needed to be trained again, we can simply pass on the saved weights so that model doesn't need to be trained from beginning

```
f = open("obj.data","a")
f.write("classes = 2\ntrain = data/train.txt\nvalid =
data/test.txt\nnames = data/obj.names\nbackup =
/mydrive/yolov4/helmet_training")
f.close()
```

- Yolov4 config file: consists of the model architecture where we can manipulate the values for hyperparameter tuning.
- First I downloaded the custom file and made changes in batch size, subdivision, filter, width and height

The updated config file can be downloaded from the link.
yolov4-custom.cfg file: link

● Zipping the files and saving in my drive

Now the data has been prepared and we have all the data we need. So select the following files:-

1. obj.names
2. obj.data
3. yolov4-custom.cfg
4. Jpg images folder
5. yolo format .txt files

Zip them and now open google drive and make a folder named **yolov4** and drag and drop the zip file.

2. Training

- Initial setup

We need to access the folders and files saved in the drive. To do that, open the google colab and run the given below command to connect with google drive.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
!ln -s /content/gdrive/My\ Drive/ /mydrive
```

- Move inside the yolov4 folder and unzip the zip file

```
%cd /mydrive/yolov4/
```

```
!unzip hard_hat_wobot_dataset.zip
```

- Cloning the AlexeyAB darknet github repo to the folder

```
!git clone https://github.com/AlexeyAB/darknet
```

- Setup/ Making of the Darknet

Training would be done using GPU. To use google colab's GPU,

  - go to runtime then change runtime type to GPU.

Now, run the given command below to set up the darknet.

```
%cd /mydrive/yolov4/darknet/
```

```
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
```

```
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile
```

```
!make
```

- Downloading yolov4 pre-trained model

```
!wget
https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137
```

- Placing the files in folders

Inside the Darknet folder empty the cfg and data folder and move the following files inside the proper folders

Inside cfg

-yolov4-custom.cfg file

Inside data folder

- obj folder(consist of .txt and .JPG training files)
- obj.names files
- obj.data file

- Splitting data

Now go inside the cfg folder and run the process.py script to split the data.

The script creates two .txt files named “train.txt” and “test.txt”.

```
import glob, os
# Current directory
current_dir = os.path.dirname(os.path.abspath(__file__))
print(current_dir)
current_dir = 'data/obj'
# Percentage of images to be used for the test set
```

```
percentage_test = 10;
# Create and/or truncate train.txt and test.txt
file_train = open('data/train.txt', 'w')
file_test = open('data/test.txt', 'w')
# Populate train.txt and test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir, "*.jpg")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))
    if counter == index_test:
        counter = 1
        file_test.write("data/obj" + "/" + title + '.jpg' + "\n")
    else:
        file_train.write("data/obj" + "/" + title + '.jpg' + "\n")
        counter = counter + 1
```

- Training the model
  - Go back to the darknet folder.
  - The darknet file is already inside the darknet folder, just pass the parameters and the model will start to train.

```
!./darknet detector train data/obj.data cfg/yolov4-custom.cfg
yolov4.conv.137 -dont_show -map
```

NOTE:

- Darknet setup is always to be done if the colab is restarted or runtime is restarted.
- If training is interrupted or stopped due to some reason, we can re-train the model from the last-trained weights saved inside the backup folder.

3. Inference

- Downloading the youtube video in drive

```
!pip3 install youtube-dl
```

```
!youtube-dl https://www.youtube.com/watch?v=G-Ms_IQFAq8
```

- Making changes in inference_yolov4-custom.cfg
  - For inference we only need a batch of size 1 and subdivision 1.
  - So make a copy of yolov4-custom.cfg file and save as inference_yolov4-custom.cfg and run the command below

```
%cd cfg
!sed -i 's/batch=64/batch=1/' inference_yolov4-custom.cfg
!sed -i 's/subdivisions=16/subdivisions=1/' inference_yolov4-custom.cfg
%cd ..
```

- Inference script

To predict the bounding boxes of the video frame, I used darknet. Since the CPU computation script takes a lot of time. So, I made a script using darknet. Since darknet is written in C language I used python bindings inside my script.
The process is as follows:-

- After the setup of Darknet, the darknet network is loaded with the given parameters like config path,weights, input video path and output video path.
- Now we read video by cv2.CaptureVideo() frame by frame.
- For each frame, the Darknet converts the frame to the darknet width and height(i.e. 416 x 416) and detects the bounding boxes using darknet.detect_images() and saves the detections.
- Detections saves label, confidence, bbox.
- Now the bbox are of darknet format so to convert back to original we use convert2original() function.
- To convert back to the original format we use bbox2points() and convert2videosize() function.
- Now we draw bounding boxes and label them with name and confidence using cv2.rectangle() and cv2.putText().
- For every helmet detection findColor() function predicts the color of helmet(explained in stage 2 evaluation)

The code is given below.

```python
import numpy as np
import time
import cv2
import os
import darknet
# from findColor import findColor
import pandas as pd

def bbox2points(bbox):
    """
    From bounding box yolo format
    to corner points cv2 rectangle
    """
    x, y, w, h = bbox
    xmin = int(round(x - (w / 2)))
    xmax = int(round(x + (w / 2)))
    ymin = int(round(y - (h / 2)))
    ymax = int(round(y + (h / 2)))
    return xmin, ymin, xmax, ymax
def convert2relative(bbox):
    """
    YOLO format use relative coordinates for annotation
    """
    x, y, w, h  = bbox
    _height     = darknet_height
    _width      = darknet_width
    return x/_width, y/_height, w/_width, h/_height


def convert2original(image, bbox):
    x, y, w, h = convert2relative(bbox)

    image_h, image_w, __ = image.shape

    orig_x       = int(x * image_w)
    orig_y       = int(y * image_h)
    orig_width   = int(w * image_w)
    orig_height  = int(h * image_h)

    bbox_converted = (orig_x, orig_y, orig_width, orig_height)
```

```
    return bbox_converted

def convert2videosize(bbox):
    x, y, w, h = bbox
    v2 = video_height/darknet_height
    v1 = video_width/darknet_width
    return(int(x*v1),int(y*v2),int(w*v1),int(h*v2))

ex_path = "../helmet/inference/"
config_file = ex_path+"inference_yolov4-custom.cfg"
weights = ex_path+"yolov4-custom_best.weights"
labels = ex_path+"obj.names"
video_file_name = ex_path + "1.mp4"
out_filename = ex_path+ "output.mp4"
data_file = ex_path+"obj.data"
thresh =0.5
#args = parser()
#check_arguments_errors(args)
network, class_names, class_colors = darknet.load_network(
        config_file,
        data_file,
        weights,
        batch_size=1
    )
darknet_width = darknet.network_width(network)
darknet_height = darknet.network_height(network)
#input_path = str2int(video_file_name)
cap = cv2.VideoCapture(video_file_name)
video_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
video_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

cap = cv2.VideoCapture(ex_path+"1.mp4")
#print(cap.get(cv2.get))
fourcc = cv2.VideoWriter_fourcc('X','V','I','D')
W = video_width
H = video_height
out = cv2.VideoWriter(ex_path+'output04.mp4', fourcc, 20,(W,H),True )
cnt = 0
while cap.isOpened():
    ret, frame =cap.read()
    if not ret:
```

```
        break
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame_resized = cv2.resize(frame_rgb, (darknet_width,
darknet_height),
                               interpolation=cv2.INTER_LINEAR)
    # frame_queue.put(frame)
    img_for_detect = darknet.make_image(darknet_width, darknet_height, 3)
    darknet.copy_image_from_bytes(img_for_detect,
frame_resized.tobytes())

    prev_time = time.time()
    detections = darknet.detect_image(network, class_names,
img_for_detect, thresh=thresh)
    # detections_queue.put(detections)
    fps = int(1/(time.time() - prev_time))
    print("FPS: {}".format(fps))# prints the average fps
    #darknet.print_detections(detections, ext_output)
    darknet.free_image(img_for_detect)

    detections_adjusted = []
    if frame is not None:
        for label, confidence, bbox in detections:
            bbox_adjusted = convert2original(frame, bbox)
            detections_adjusted.append((label, confidence,
bbox_adjusted))
            bbox = bbox2points(bbox)
            left, top, right, bottom =convert2videosize(bbox)
            print(bbox)
            bbox_color = (0,255,0)if(label=="head") else
findColor(frame,[left,top,right,bottom])
            cv2.rectangle(frame, (left, top), (right, bottom),bbox_color
, 2)
            cv2.putText(frame, "{} [{:.2f}]".format(label,
float(confidence)),
                        (left, top - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                        bbox_color, 2)
    out.write(frame)
    print(cnt)
    cnt+=1
out.release()
cap.release()
```

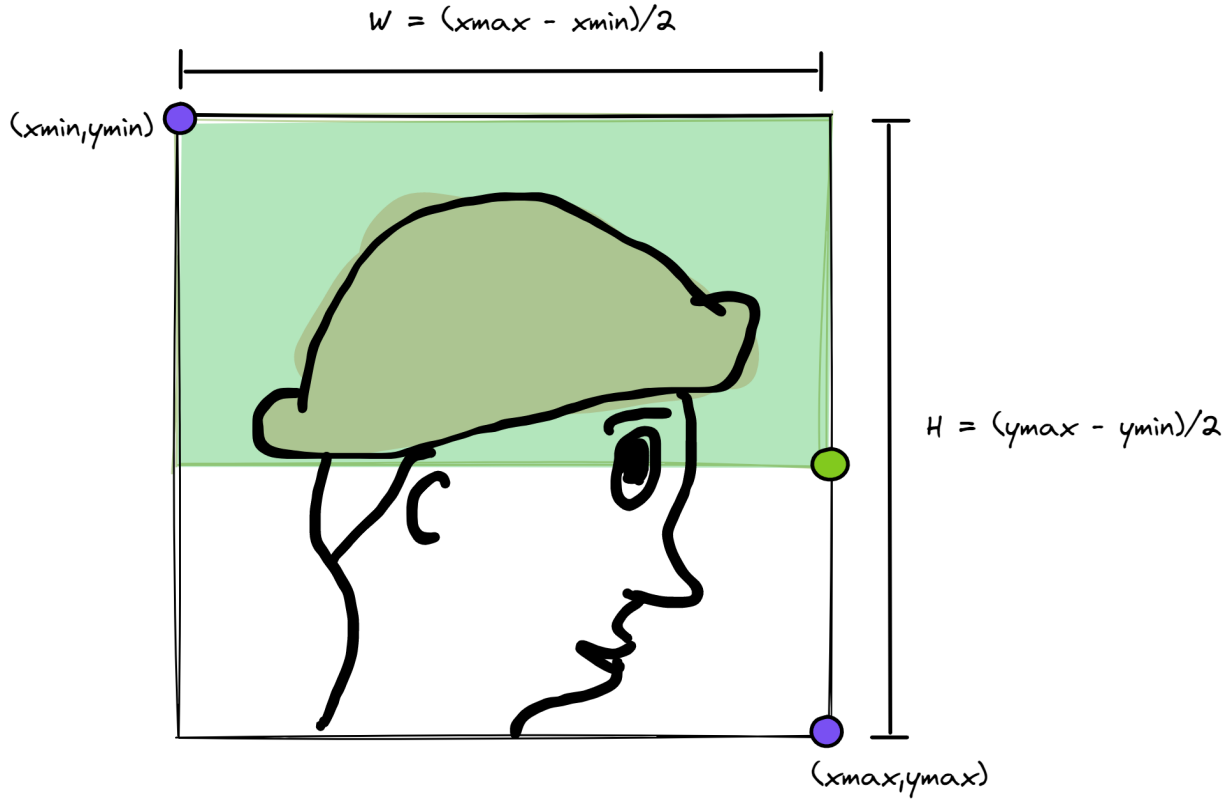
```
cv2.destroyAllWindows()
```

- After video is been made

The output file does not play in google chrome because google chrome only plays H264 videos only. So after the video is made, run the ffmpeg command to convert the output.mp4 to new output.mp4 so that the video is playable in the browser.

```
!ffmpeg -i output123.mp4 -vcodec h264 output1234.mp4
```

Stage 2 Evaluation



Task:

To find the color of the bounding box

My Approach:

The predicted bounding boxes that we got after inferencing the video have the coordinates xmin, ymin, xmax and ymax(marked in the diagram).
From observation of the image, we can see that the helmet part covers 95% of the top half of the box (marked in green shaded region). So to find the color of the helmet we first need to find the n_xmax and n_ ymax.
Using cartesian coordinate system,

$$n\_xmax = xmax$$
$$n\_ymax = (ymin + ymax)/2 \ (taking\ the\ rounded\ value)$$

Now by applying KMeans clustering(grouping of similar data points) we can find the color of the helmet.

The function findColor() is given below

```
import cv2
from sklearn.cluster import KMeans

def findColor(frame,bbox):
    '''

    Function to find the color of the hard_hat_workers

    input:   filename,predicted bounding box coordinate
    process: Using KMeans value find the color of the hat
    returns: color in BGR format

    '''
    bbox = [max(0,i) for i in bbox]
    x1,y1,x2,y2 = bbox[0], bbox[1], bbox[-2], bbox[-1]
    new_y2 = (y2+y1)//2
    src_o = frame
    src = src_o[y1:new_y2,x1:x2]
    clt = KMeans(n_clusters=1)#to find the maximum value of the cluster
    clt.fit(src.reshape(-1, 3))
    color = clt.cluster_centers_.tolist()[0]
    return(color)
```

findColor() function takes two parameters:- frame(or image) and bounding box coordinates and returns the approximate value of the highest color found in the top half of the box.

Why KMeans?

The objective of K-means is to group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (*k*) of clusters in a dataset. A cluster refers to a collection of data points aggregated together because of certain similarities.

Stage 3 Evaluation

Task: Inference the given 250 test images and save the coordinates in the form of pascal voc xml format.

My Approach:

To make pascal voc. xml files, first we need data i.e bounding box, filename etc. The inference script given below is used to detect bounding boxes from the test image. The detected bounding boxes with other details like image filename, width, height,depth of image and class(label name) is saved inside a DataFrame which is hence passed inside csv_to_xml() function.
There are some images where there is no detection of bounding boxes for which a new_df is created inside the else function and new_df is passed inside csv_to_xml() function.

```
import numpy as np
import time
import cv2
import os
import pandas as pd
from csvToXml import csv_to_xml

CONFIDENCE_THRESHOLD=0.5

CONFIG_FILE = "inference_yolov4-custom.cfg"
WEIGHTS_FILE = "yolov4-custom_best.weights"
LABELS_FILE = "obj.names"

LABELS = open(LABELS_FILE).read().strip().split("\n")
net = cv2.dnn.readNetFromDarknet(CONFIG_FILE, WEIGHTS_FILE)

img_path = "Images/"
save_path = "annotations/"

for file in os.listdir(img_path):
   if(file[:-3]+"xml" not in os.listdir(save_path)):
```

```
        INPUT_FILE = img_path+file
        image = cv2.imread(INPUT_FILE)
        (W, H, C) = image.shape
        df =
pd.DataFrame(columns=("filename","class","xmin","ymin","xmax","ymax","wi
dth","height","depth"))
        # determine only the *output* layer names that we need from YOLO
        ln = net.getLayerNames()
        ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
        blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416),
            swapRB=True, crop=False)
        net.setInput(blob)
        start = time.time()
        layerOutputs = net.forward(ln)
        end = time.time()


        print("[INFO] YOLO took {:.6f} seconds".format(end - start))


        # initialize our lists of detected bounding boxes, confidences,
and
        # class IDs, respectively
        boxes = []
        confidences = []
        classIDs = []

        # loop over each of the layer outputs
        for output in layerOutputs:
            # loop over each of the detections
            for detection in output:
                # extract the class ID and confidence (i.e., probability)
of
                # the current object detection
                scores = detection[5:]
                classID = np.argmax(scores)
                confidence = scores[classID]

                # filter out weak predictions by ensuring the detected
                # probability is greater than the minimum probability
                if confidence > CONFIDENCE_THRESHOLD:
```

```
                    # scale the bounding box coordinates back relative to
the
                    # size of the image, keeping in mind that YOLO
actually
                    # returns the center (x, y)-coordinates of the
bounding
                    # box followed by the boxes' width and height
                    box = detection[0:4] * np.array([W, H, W, H])
                    (centerX, centerY, width, height) = box.astype("int")

                    # use the center (x, y)-coordinates to derive the top
and
                    # and left corner of the bounding box
                    x = int(centerX - (width / 2))
                    y = int(centerY - (height / 2))

                    # update our list of bounding box coordinates,
confidences,
                    # and class IDs
                    boxes.append([x, y, int(width), int(height)])
                    confidences.append(float(confidence))
                    classIDs.append(classID)

        # apply non-maxima suppression to suppress weak, overlapping
bounding
        # boxes
        idxs = cv2.dnn.NMSBoxes(boxes, confidences, CONFIDENCE_THRESHOLD,
            CONFIDENCE_THRESHOLD)



        # ensure at least one detection exists
        if len(idxs) > 0:
            # loop over the indexes we are keeping
            for i in idxs.flatten():
                # extract the bounding box coordinates
                (x, y) = (boxes[i][0], boxes[i][1])
                (w, h) = (boxes[i][2], boxes[i][3])

                xmin,ymin,xmax,ymax = x, y, x+w, y+h
                df =
```

```
df.append({"filename":file,"class":LABELS[classIDs[i]],
"xmin":xmin,"ymin":ymin,"xmax":xmax,"ymax":ymax,"width":W,"height":H,
"depth":C}, ignore_index=True)

        else:
            new_df =
pd.DataFrame(columns=("filename","width","height","depth"))
            new_df = new_df.append({"filename":file,"width":W,"height":H,
"depth":C}, ignore_index=True)
            df = new_df
        csv_to_xml(df, img_path, save_path)
    print(len(os.listdir(save_path)))
```

By using xml.etree.ElementTree package we can create the desired xml files.
The function csv_to_xml() takes to 3 parameters

- Df: the data and bounding boxes of the image
- Image path
- Save path: where the xml files are to be saved

The function work as follows:-

- Initial root element is created i.e annotation.
- Then a subelement is created i.e folder,filename, size,segmented)
- If the df is of length 4, which means there is no bndbox, hence it doesn't pass through the for loop. Else for loop will be executed.
- Now the xml files have been created but they do not have indentation or say they do not look pretty. So to make them pretty ET.indent() function is used.
- Now the tree is written with the same image filename and saved inside the save_path.

```
# import pandas as pd
import xml.etree.ElementTree as ET

def csv_to_xml(df,img_path,save_path):

    # we make root element
    annotation = ET.Element("annotation")

    # create sub element
```

```
    folder = ET.SubElement(annotation, "folder")
    folder.text = "images"
    filename = ET.SubElement(annotation, "filename")
    filename.text = df['filename'][0]
    size = ET.SubElement(annotation, "size")
    ET.SubElement(size, "width").text = str(df['width'][0])
    ET.SubElement(size, "height").text = str(df['height'][0])
    ET.SubElement(size, "depth").text = str(df["depth"][0])
    ET.SubElement(annotation,"segmented").text = "0"
    if(len(df)>4):
        for i in range(len(df)):
                obj = ET.SubElement(annotation,"object")
                ET.SubElement(obj,"name").text = df['class'][i]
                ET.SubElement(obj,"pose").text = "Unspecified"
                ET.SubElement(obj,"truncated").text = "0"
                ET.SubElement(obj,"occluded").text = "0"
                ET.SubElement(obj,"difficult").text = "0"
                bndbox = ET.SubElement(obj, "bndbox")
                ET.SubElement(bndbox, "xmin").text = str(df['xmin'][i])
                ET.SubElement(bndbox, "ymin").text = str(df['ymin'][i])
                ET.SubElement(bndbox, "xmax").text = str(df['xmax'][i])
                ET.SubElement(bndbox, "ymax").text = str(df['ymax'][i])

    tree = ET.ElementTree(annotation)
    ET.indent(tree,space="    ")# to make pretty xml file

    # write the tree into an XML file
    tree.write(save_path+filename.text[:-3]+"xml", encoding ='utf-8',
xml_declaration = False)
```

NOTE:

- ET.indent() only works in the latest python version i.e version 3.9. So, the latest python is needed to be installed for the above function to work.
- Inference script requires cv2 of version 4.4 or above.