```python
#!/usr/bin/env python
# coding: utf-8
import cv2
import numpy as np
import os
import yaml
from yaml.loader import SafeLoader


class YOLO_Pred():
    def __init__(self,onnx_model,data_yaml):
        # load YAML
        with open(data_yaml,mode='r') as f:
            data_yaml = yaml.load(f,Loader=SafeLoader)

        self.labels = data_yaml['names']
        self.nc = data_yaml['nc']

        # load YOLO model
        self.yolo = cv2.dnn.readNetFromONNX(onnx_model)
        self.yolo.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
        self.yolo.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)



    def predictions(self,image):

        row, col, d = image.shape
        # get the YOLO prediction from the the image
        # step-1 convert image into square image (array)
        max_rc = max(row,col)
        input_image = np.zeros((max_rc,max_rc,3),dtype=np.uint8)
        input_image[0:row,0:col] = image
        # step-2: get prediction from square array
        INPUT_WH_YOLO = 640
        blob = cv2.dnn.blobFromImage(input_image,1/255,
(INPUT_WH_YOLO,INPUT_WH_YOLO),swapRB=True,crop=False)
        self.yolo.setInput(blob)
        preds = self.yolo.forward() # detection or prediction from YOLO

        # Non Maximum Supression
        # step-1: filter detection based on confidence (0.4) and probability score (0.25)
        detections = preds[0]
        boxes = []
        confidences = []
        classes = []

        # widht and height of the image (input_image)
        image_w, image_h = input_image.shape[:2]
        x_factor = image_w/INPUT_WH_YOLO
        y_factor = image_h/INPUT_WH_YOLO

        for i in range(len(detections)):
            row = detections[i]
            confidence = row[4] # confidence of detection an object
            if confidence > 0.4:
                class_score = row[5:].max() # maximum probability from 20 objects
                class_id = row[5:].argmax() # get the index position at which max probabilty
occur

                if class_score > 0.25:
                    cx, cy, w, h = row[0:4]
                    # construct bounding from four values
                    # left, top, width and height
                    left = int((cx - 0.5*w)*x_factor)
                    top = int((cy - 0.5*h)*y_factor)
                    width = int(w*x_factor)
```

```
                height = int(h*y_factor)

                box = np.array([left,top,width,height])

                # append values into the list
                confidences.append(confidence)
                boxes.append(box)
                classes.append(class_id)

    # clean
    boxes_np = np.array(boxes).tolist()
    confidences_np = np.array(confidences).tolist()

    # NMS
    index = cv2.dnn.NMSBoxes(boxes_np,confidences_np,0.25,0.45).flatten()


    # Draw the Bounding
    for ind in index:
        # extract bounding box
        x,y,w,h = boxes_np[ind]
        bb_conf = int(confidences_np[ind]*100)
        classes_id = classes[ind]
        class_name = self.labels[classes_id]
        colors = self.generate_colors(classes_id)

        text = f'{class_name}: {bb_conf}%'

        cv2.rectangle(image,(x,y),(x+w,y+h),colors,2)
        cv2.rectangle(image,(x,y-30),(x+w,y),colors,-1)

        cv2.putText(image,text,(x,y-10),cv2.FONT_HERSHEY_PLAIN,0.7,(0,0,0),1)


    return image


def generate_colors(self,ID):
    np.random.seed(10)
    colors = np.random.randint(100,255,size=(self.nc,3)).tolist()
    return tuple(colors[ID])
```