inmind.ai

# Machine Learning Track

## Computer Vision Final Project

**Presented to:**

**Elio Hanna – Dani Azzam – Roy Daher – Ziad Saoud**

**Presented by:**

**Mohamad Jaber**

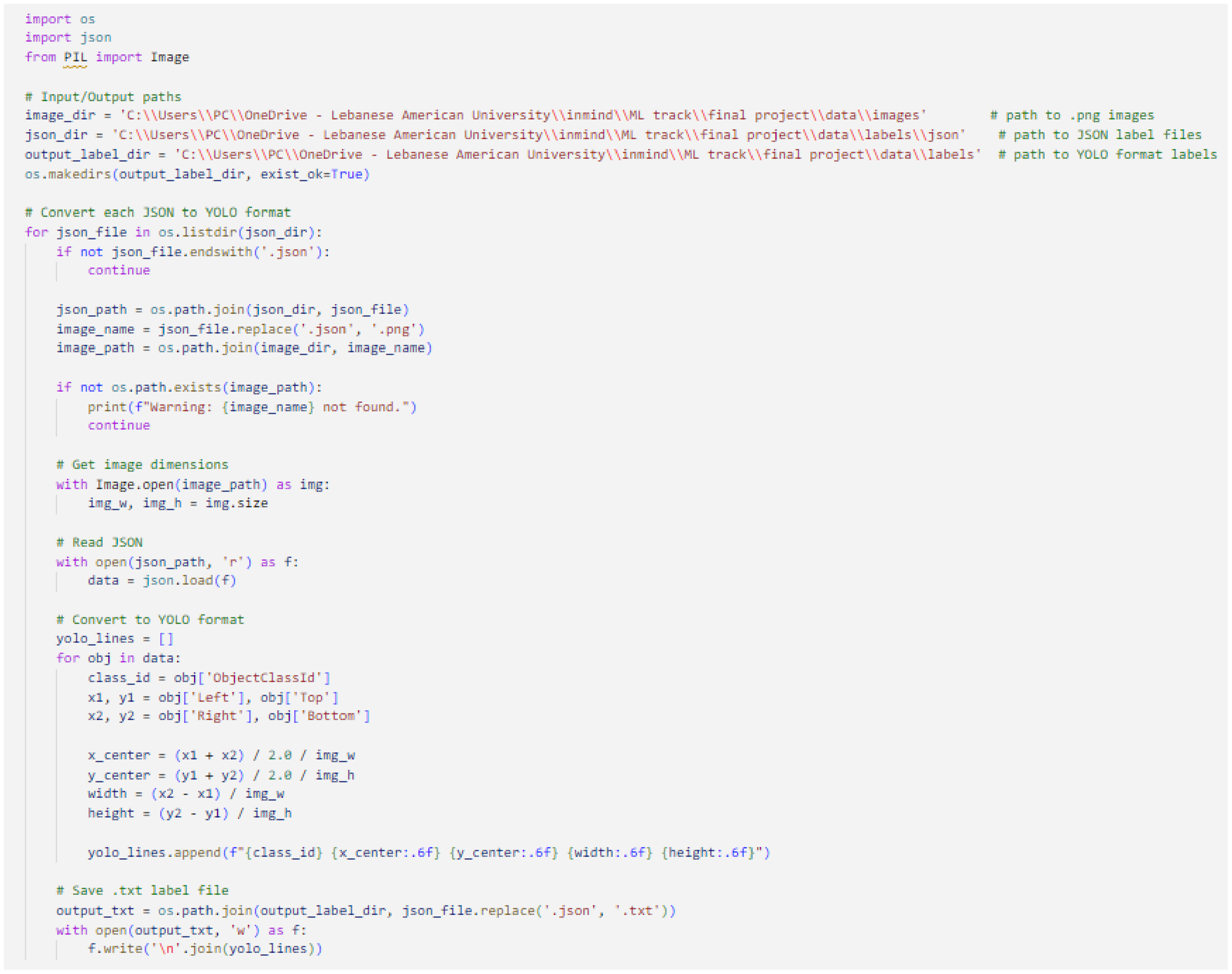
**Date:**

**Wednesday, April 2, 2025**

## Table of Contents

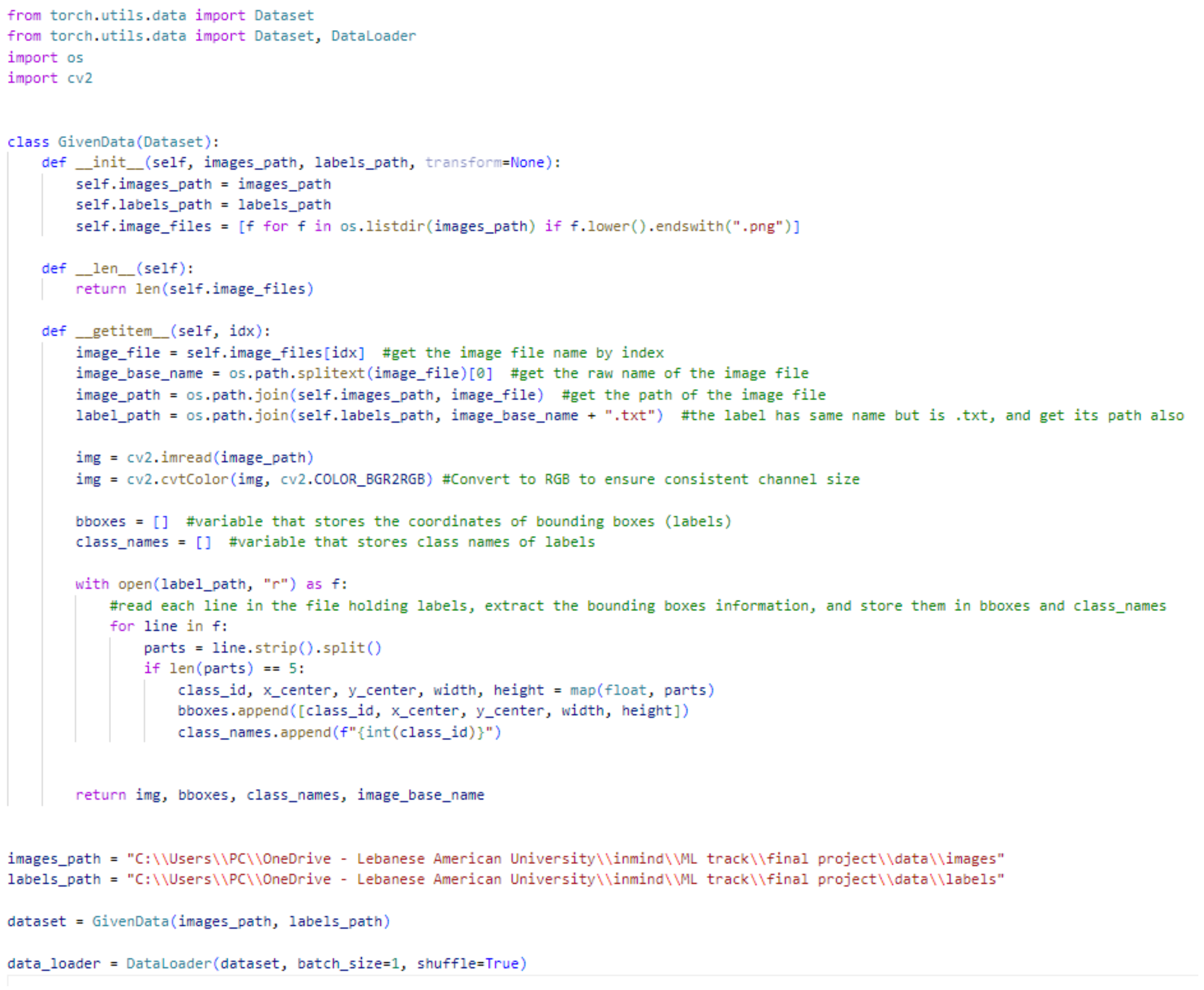
## Data Preparation & Visualization:

### 1. Converting from JSON format to txt format



```python
import os
import json
from PIL import Image

# Input/Output paths
image_dir = 'C:\\Users\\PC\\OneDrive - Lebanese American University\\inmind\\ML track\\final project\\data\\images'        # path to .png images
json_dir = 'C:\\Users\\PC\\OneDrive - Lebanese American University\\inmind\\ML track\\final project\\data\\labels\\json'    # path to JSON label files
output_label_dir = 'C:\\Users\\PC\\OneDrive - Lebanese American University\\inmind\\ML track\\final project\\data\\labels'  # path to YOLO format labels
os.makedirs(output_label_dir, exist_ok=True)

# Convert each JSON to YOLO format
for json_file in os.listdir(json_dir):
    if not json_file.endswith('.json'):
        continue

    json_path = os.path.join(json_dir, json_file)
    image_name = json_file.replace('.json', '.png')
    image_path = os.path.join(image_dir, image_name)

    if not os.path.exists(image_path):
        print(f"Warning: {image_name} not found.")
        continue

    # Get image dimensions
    with Image.open(image_path) as img:
        img_w, img_h = img.size

    # Read JSON
    with open(json_path, 'r') as f:
        data = json.load(f)

    # Convert to YOLO format
    yolo_lines = []
    for obj in data:
        class_id = obj['ObjectClassId']
        x1, y1 = obj['Left'], obj['Top']
        x2, y2 = obj['Right'], obj['Bottom']

        x_center = (x1 + x2) / 2.0 / img_w
        y_center = (y1 + y2) / 2.0 / img_h
        width = (x2 - x1) / img_w
        height = (y2 - y1) / img_h

        yolo_lines.append(f"{class_id} {x_center:.6f} {y_center:.6f} {width:.6f} {height:.6f}")

    # Save .txt label file
    output_txt = os.path.join(output_label_dir, json_file.replace('.json', '.txt'))
    with open(output_txt, 'w') as f:
        f.write('\n'.join(yolo_lines))
```

## 2. Data Loader:



```python
from torch.utils.data import Dataset
from torch.utils.data import Dataset, DataLoader
import os
import cv2


class GivenData(Dataset):
    def __init__(self, images_path, labels_path, transform=None):
        self.images_path = images_path
        self.labels_path = labels_path
        self.image_files = [f for f in os.listdir(images_path) if f.lower().endswith(".png")]

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

    def __getitem__(self, idx):
        image_file = self.image_files[idx]  #get the image file name by index
        image_base_name = os.path.splitext(image_file)[0]  #get the raw name of the image file
        image_path = os.path.join(self.images_path, image_file)  #get the path of the image file
        label_path = os.path.join(self.labels_path, image_base_name + ".txt")  #the label has same name but is .txt, and get its path also

        img = cv2.imread(image_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #Convert to RGB to ensure consistent channel size

        bboxes = []  #variable that stores the coordinates of bounding boxes (labels)
        class_names = []  #variable that stores class names of labels

        with open(label_path, "r") as f:
            #read each line in the file holding labels, extract the bounding boxes information, and store them in bboxes and class_names
            for line in f:
                parts = line.strip().split()
                if len(parts) == 5:
                    class_id, x_center, y_center, width, height = map(float, parts)
                    bboxes.append([class_id, x_center, y_center, width, height])
                    class_names.append(f"{int(class_id)}")


        return img, bboxes, class_names, image_base_name


images_path = "C:\\Users\\PC\\OneDrive - Lebanese American University\\inmind\\ML track\\final project\\data\\images"
labels_path = "C:\\Users\\PC\\OneDrive - Lebanese American University\\inmind\\ML track\\final project\\data\\labels"

dataset = GivenData(images_path, labels_path)

data_loader = DataLoader(dataset, batch_size=1, shuffle=True)
```

## 3. *Function to visualize labeled images:*

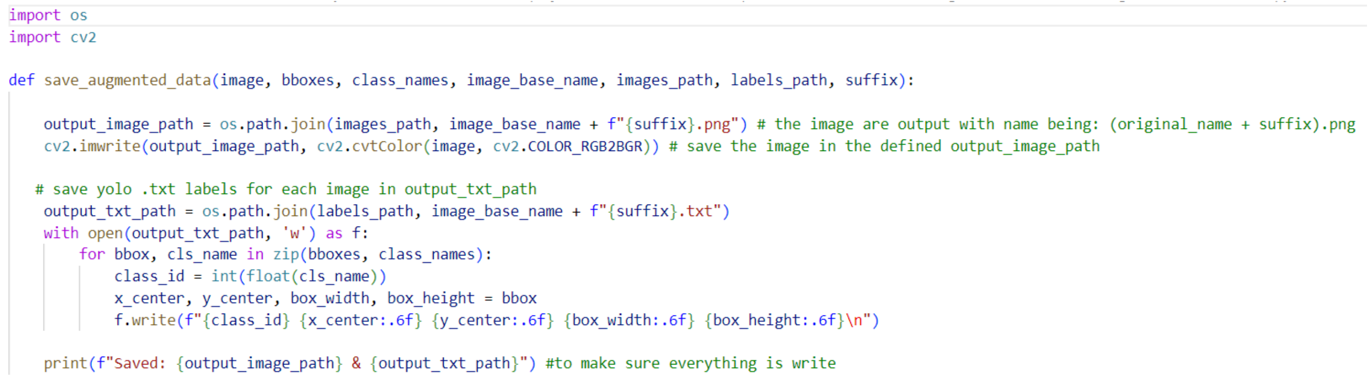
```python
import cv2
import matplotlib.pyplot as plt

#the function displays the image with the bounding boxes(s)
def visualize(image, bboxes, class_names, box_color=(255, 0, 0), thickness=2):

    img = image.copy()
    img_h, img_w = img.shape[:2] #get height and width

    for bbox, class_name in zip(bboxes, class_names):
        x_center, y_center, w, h = bbox  #retrieve the coordinates of bounding box from bbox variable

        #convert normalized yolo format to pixel coordinates
        x_min = int((x_center - w / 2) * img_w)
        y_min = int((y_center - h / 2) * img_h)
        x_max = int((x_center + w / 2) * img_w)
        y_max = int((y_center + h / 2) * img_h)

        #draw bounding box
        cv2.rectangle(img, (x_min, y_min), (x_max, y_max), color=box_color, thickness=thickness)

        #draw class label background
        ((text_width, text_height), _) = cv2.getTextSize(str(class_name), cv2.FONT_HERSHEY_SIMPLEX, 0.5, 2)
        cv2.rectangle(img, (x_min, y_min - text_height - 5), (x_min + text_width, y_min), box_color, -1)

        #draw class label text
        cv2.putText(
            img,
            text=str(class_name),
            org=(x_min, y_min - 5),
            fontFace=cv2.FONT_HERSHEY_SIMPLEX,
            fontScale=0.5,
            color=(0, 255, 0),
            thickness=1,
            lineType=cv2.LINE_AA,
        )

    # Show image
    plt.figure(figsize=(12, 12))
    plt.axis('off')
    plt.imshow(img)
    plt.show()
```

## 4. *Augmentation:*

### - Defining the Pipeline:

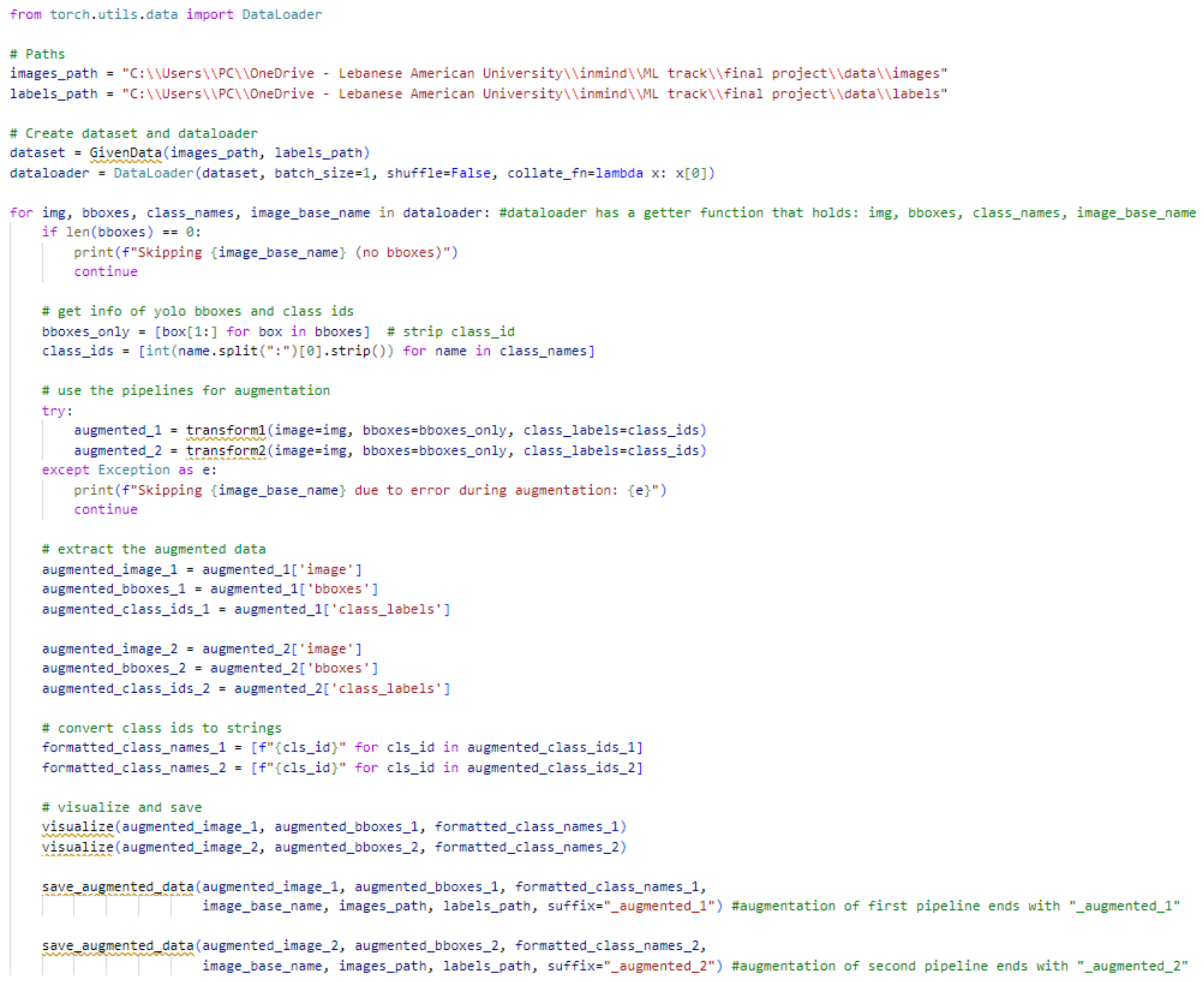
```python
import albumentations as A
import cv2

#define 2 different pipelines for image augmentation

transform1 = A.Compose([
    A.Blur(blur_limit=3, p=1.0), #add bluring to image
    A.Rotate(limit=30, border_mode=cv2.BORDER_CONSTANT, p=0.5),  # Rotation within -30 and +30 deg
], bbox_params=A.BboxParams(format='yolo', label_fields=["class_labels"])) #show the bboxes based on yolo format

transform2 = A.Compose([
    A.GaussNoise(var_limit=(20.0, 100.0), p=1.0) #add gaussian noise to image
], bbox_params=A.BboxParams(format='yolo', label_fields=["class_labels"])) #show the bboxes based on yolo format
```

- ## Function to save data with their labels:



```python
import os
import cv2

def save_augmented_data(image, bboxes, class_names, image_base_name, images_path, labels_path, suffix):

    output_image_path = os.path.join(images_path, image_base_name + f"{suffix}.png") # the image are output with name being: (original_name + suffix).png
    cv2.imwrite(output_image_path, cv2.cvtColor(image, cv2.COLOR_RGB2BGR)) # save the image in the defined output_image_path

   # save yolo .txt labels for each image in output_txt_path
    output_txt_path = os.path.join(labels_path, image_base_name + f"{suffix}.txt")
    with open(output_txt_path, 'w') as f:
        for bbox, cls_name in zip(bboxes, class_names):
            class_id = int(float(cls_name))
            x_center, y_center, box_width, box_height = bbox
            f.write(f"{class_id} {x_center:.6f} {y_center:.6f} {box_width:.6f} {box_height:.6f}\n")

    print(f"Saved: {output_image_path} & {output_txt_path}") #to make sure everything is write
```

- ## Augmenting Images and saving them with their labels:



```python
from torch.utils.data import DataLoader

# Paths
images_path = "C:\\Users\\PC\\OneDrive - Lebanese American University\\inmind\\ML track\\final project\\data\\images"
labels_path = "C:\\Users\\PC\\OneDrive - Lebanese American University\\inmind\\ML track\\final project\\data\\labels"

# Create dataset and dataloader
dataset = GivenData(images_path, labels_path)
dataloader = DataLoader(dataset, batch_size=1, shuffle=False, collate_fn=lambda x: x[0])

for img, bboxes, class_names, image_base_name in dataloader: #dataloader has a getter function that holds: img, bboxes, class_names, image_base_name
    if len(bboxes) == 0:
        print(f"Skipping {image_base_name} (no bboxes)")
        continue

    # get info of yolo bboxes and class ids
    bboxes_only = [box[1:] for box in bboxes]  # strip class_id
    class_ids = [int(name.split(":")[0].strip()) for name in class_names]

    # use the pipelines for augmentation
    try:
        augmented_1 = transform1(image=img, bboxes=bboxes_only, class_labels=class_ids)
        augmented_2 = transform2(image=img, bboxes=bboxes_only, class_labels=class_ids)
    except Exception as e:
        print(f"Skipping {image_base_name} due to error during augmentation: {e}")
        continue

    # extract the augmented data
    augmented_image_1 = augmented_1['image']
    augmented_bboxes_1 = augmented_1['bboxes']
    augmented_class_ids_1 = augmented_1['class_labels']

    augmented_image_2 = augmented_2['image']
    augmented_bboxes_2 = augmented_2['bboxes']
    augmented_class_ids_2 = augmented_2['class_labels']

    # convert class ids to strings
    formatted_class_names_1 = [f"{cls_id}" for cls_id in augmented_class_ids_1]
    formatted_class_names_2 = [f"{cls_id}" for cls_id in augmented_class_ids_2]

    # visualize and save
    visualize(augmented_image_1, augmented_bboxes_1, formatted_class_names_1)
    visualize(augmented_image_2, augmented_bboxes_2, formatted_class_names_2)

    save_augmented_data(augmented_image_1, augmented_bboxes_1, formatted_class_names_1,
                        image_base_name, images_path, labels_path, suffix="_augmented_1") #augmentation of first pipeline ends with "_augmented_1"

    save_augmented_data(augmented_image_2, augmented_bboxes_2, formatted_class_names_2,
                        image_base_name, images_path, labels_path, suffix="_augmented_2") #augmentation of second pipeline ends with "_augmented_2"
```

## 5. *Splitting the data:*

### - Fixing class ids:

```python
import os

labels_root = "C:/Users/PC/OneDrive - Lebanese American University/inmind/ML track/final project/data/labels"

for filename in os.listdir(labels_root):
    if filename.endswith('.txt'):
        file_path = os.path.join(labels_root, filename)
        with open(file_path, 'r') as f:
            lines = f.readlines()

        new_lines = [] #create a list that holds the updated lines
        for line in lines:
            parts = line.strip().split()
            if len(parts) == 5:
                class_id = int(parts[0]) - 1  # subtract the class ids by 1 as to have class ids starting from 0 till 4, instead of 1 till 5
                new_line = f"{class_id} {' '.join(parts[1:])}\n" # the new_line holds the same content as old one, but class id is subtracted by 1
                new_lines.append(new_line) # append the new line or lines for each label file in new_lines list

        with open(file_path, 'w') as f:
            f.writelines(new_lines)  # erase all the old content from the old label file, and write the new lines on it
```

### - New class ids:

```json
[
    {
        "Id": 0,
        "Name": "tugger",
        "ColorCode": "bc3ca0",
        "ParentObjectClassId": null
    },
    {
        "Id": 1,
        "Name": "cabinet",
        "ColorCode": "c45e69",
        "ParentObjectClassId": null
    },
    {
        "Id": 2,
        "Name": "str",
        "ColorCode": "33bf2d",
        "ParentObjectClassId": null
    },
    {
        "Id": 3,
        "Name": "box",
        "ColorCode": "164ab0",
        "ParentObjectClassId": null
    },
    {
        "Id": 4,
        "Name": "forklift",
        "ColorCode": "2eb3d5",
        "ParentObjectClassId": null
    }
]
```

- Splitting the data:

```python
import os
import shutil
import random

# source folder holding both images and labels
source_dir = "C:\\Users\\PC\\OneDrive - Lebanese American University\\inmind\\ML track\\final project\\data"

# output path holding the split data
output_base = "C:\\Users\\PC\\OneDrive - Lebanese American University\\inmind\\ML track\\final project\\data\\yolov5_split"

# we are expected to split between train and validation data
def split_dataset(source_dir, output_base = output_base, split_ratio=0.8): # 80% train and 20% validation/test

    #fix the organization of folders as to match that accepted by yolo
    train_dir = os.path.join(output_base, 'train')
    val_dir = os.path.join(output_base, 'val')

    os.makedirs(os.path.join(train_dir, 'images'), exist_ok=True)
    os.makedirs(os.path.join(train_dir, 'labels'), exist_ok=True)
    os.makedirs(os.path.join(val_dir, 'images'), exist_ok=True)
    os.makedirs(os.path.join(val_dir, 'labels'), exist_ok=True)

    #shuffle then split the data
    image_list = [f for f in os.listdir(os.path.join(source_dir, 'images')) if f.endswith('.png')]
    random.shuffle(image_list)
    split_point = int(len(image_list) * split_ratio)

    for i, img_name in enumerate(image_list):
        target_dir = train_dir if i < split_point else val_dir
        shutil.copy(
            os.path.join(source_dir, 'images', img_name),
            os.path.join(target_dir, 'images', img_name)
        )
        label_name = img_name.replace('.png', '.txt')
        shutil.copy(
            os.path.join(source_dir, 'labels', label_name),
            os.path.join(target_dir, 'labels', label_name)
        )

    return train_dir, val_dir

train_dir, val_dir = split_dataset(source_dir, split_ratio=0.8)
```

## Model Training & Evaluation:

### 1. Cloning yolo:

```
%cd /content
!git clone https://github.com/ultralytics/yolov5
%cd yolov5
!pip install -r requirements.txt
```

### 2. Creating yaml file:

```
#In case I am using VS Code those are the paths for train and val
train: C:/Users/PC/OneDrive - Lebanese American University/inmind/ML track/final project/data/yolov5_split/train/images
val: C:/Users/PC/OneDrive - Lebanese American University/inmind/ML track/final project/data/yolov5_split/val/images

nc: 5
names: ['tugger', 'cabinet', 'str', 'box', 'forklift']

#-----------------------------------------------------------------------------------------------------------------
#In case I am using colab those are the paths for train and val
# train: content/yolov5_split/train/images
# val: content/yolov5_split/val/images

# nc: 5
# names: ['tugger', 'cabinet', 'str', 'box', 'forklift']
```

### 3. First Train:

Go to directory of yolov5

```
%cd /content/yolov5
```

Train the model

```
!python train.py --data "/content/yolov5/data.yaml" --weights yolov5s.pt --epochs 50 --batch-size 8 --name first_run
```

Test the first trained model

```
!python val.py --data "/content/yolov5/data.yaml" --weights runs/train/first_run2/weights/best.pt --task val
```

Apply inference on the first trained model

```
!python detect.py --weights runs/train/first_run2/weights/best.pt --img 640 --conf 0.25 --source "/content/yolov5_split/val/images" --save-txt
```

## 4. Second Train:

Train the second model with updated hyperparameters

```
!python train.py --data data.yaml --weights runs/train/first_run2/weights/best.pt --cfg models/yolov5s.yaml --epochs 100 --batch-size 12 --img 640 --name tunned_run
```

Test the second trained model

```
!python val.py --data "/content/yolov5/data.yaml" --weights runs/train/tunned_run/weights/best.pt --task val --img 640
```

Apply inference using the 2nd trained model

```
!python detect.py --weights runs/train/tunned_run/weights/best.pt --img 640 --conf 0.25 --source "/content/yolov5_split/val/images" --save-txt
```

## 5. Try an inference example:

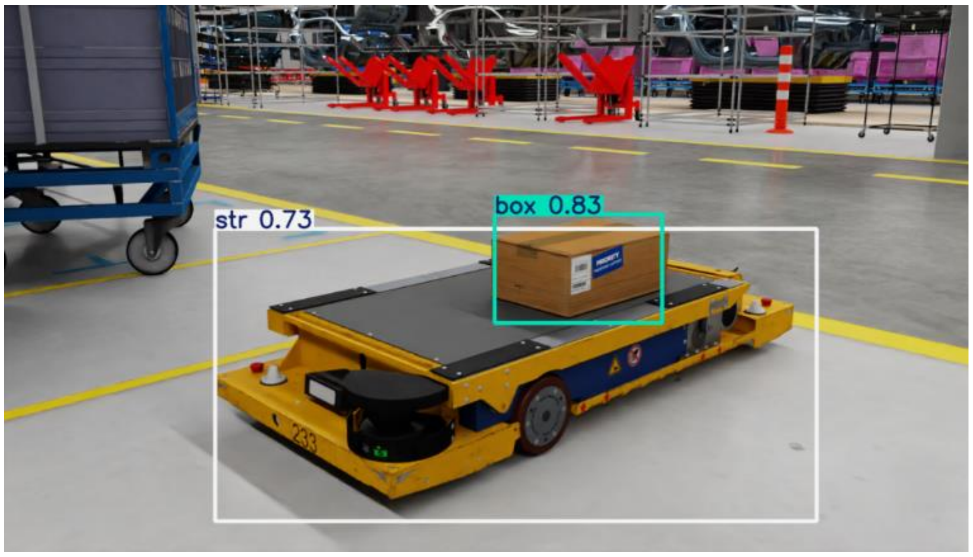
### Check which files did we apply inference on

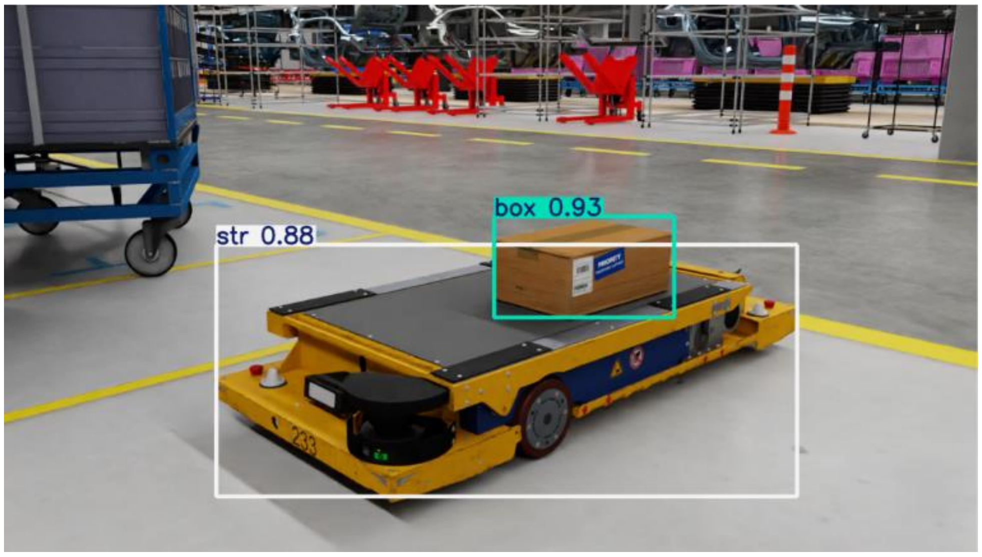
```
import os

for root, dirs, files in os.walk("runs/detect"):
    for file in files:
        print(os.path.join(root, file))
```

### Display the inference of one image using both models

```
from IPython.display import Image, display
display(Image(filename='runs/detect/exp/69.png'))
```

```
display(Image(filename='runs/detect/exp2/69.png'))
```


str 0.73
box 0.83


str 0.88
box 0.93

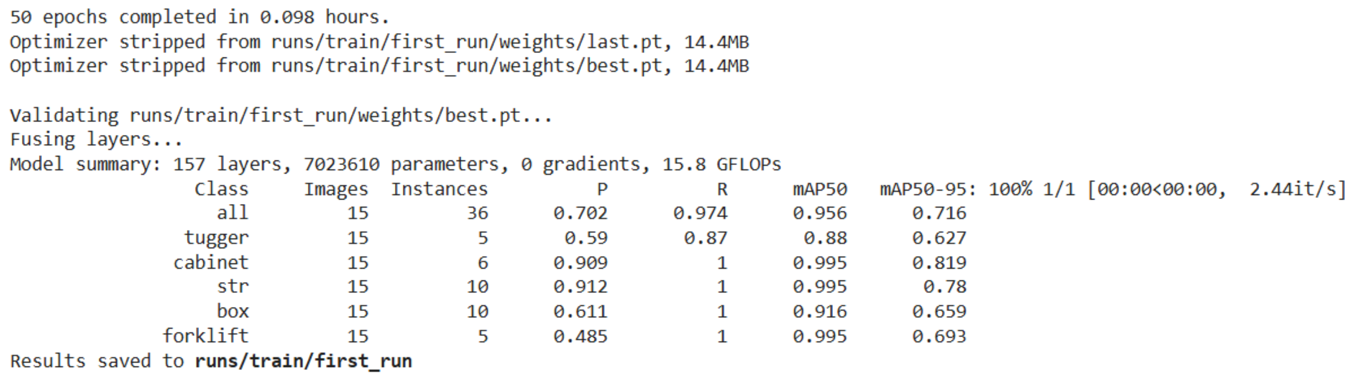
## 6. *Tensorboard + Evaluation:*

### - Hyper-parameters:

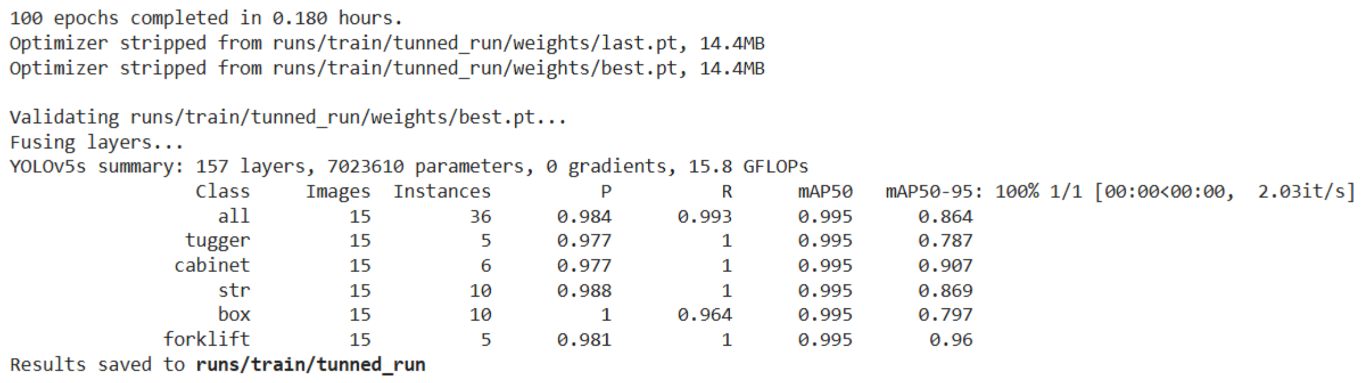
The first model uses 50 epochs and batch size = 8

The second model uses 100 epochs and batch size = 12

### - Metrics Overview:



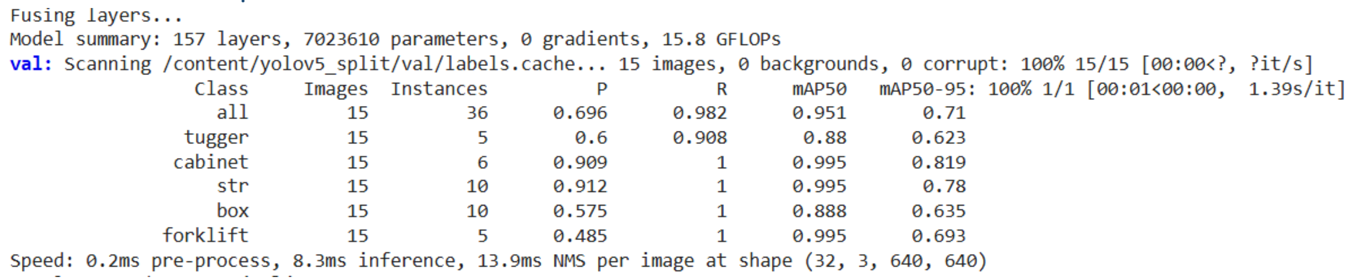
```
50 epochs completed in 0.098 hours.
Optimizer stripped from runs/train/first_run/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/first_run/weights/best.pt, 14.4MB

Validating runs/train/first_run/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  2.44it/s]
                   all         15         36      0.702      0.974      0.956      0.716
                tugger         15          5       0.59       0.87       0.88      0.627
               cabinet         15          6      0.909          1      0.995      0.819
                   str         15         10      0.912          1      0.995       0.78
                   box         15         10      0.611          1      0.916      0.659
              forklift         15          5      0.485          1      0.995      0.693
Results saved to runs/train/first_run
```



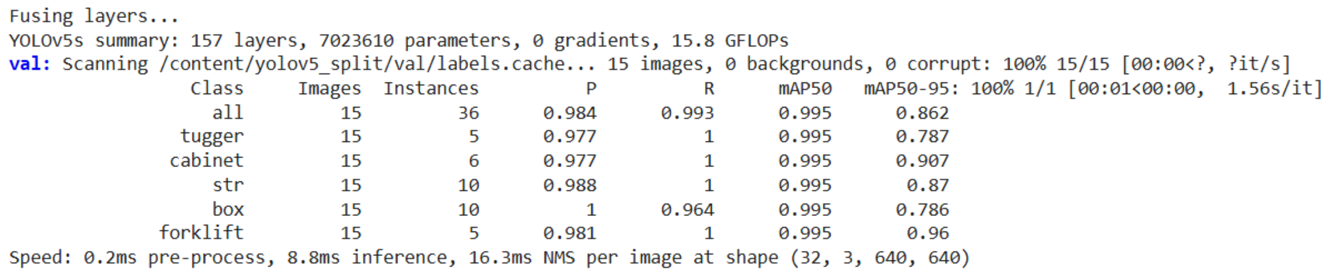
```
100 epochs completed in 0.180 hours.
Optimizer stripped from runs/train/tunned_run/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/tunned_run/weights/best.pt, 14.4MB

Validating runs/train/tunned_run/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  2.03it/s]
                   all         15         36      0.984      0.993      0.995      0.864
                tugger         15          5      0.977          1      0.995      0.787
               cabinet         15          6      0.977          1      0.995      0.907
                   str         15         10      0.988          1      0.995      0.869
                   box         15         10          1      0.964      0.995      0.797
              forklift         15          5      0.981          1      0.995       0.96
Results saved to runs/train/tunned_run
```

It can seen clearly how the metrics have improved from the first train to the second train. This is logical since the second train is a fine tunned version of the first, and also the second has higher chances of learning from the data due to having 100 epochs > 50 epochs.

- Inference Speed:



```
Fusing layers...
Model summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs
val: Scanning /content/yolov5_split/val/labels.cache... 15 images, 0 backgrounds, 0 corrupt: 100% 15/15 [00:00<?, ?it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:01<00:00,  1.39s/it]
                   all         15         36      0.696      0.982      0.951       0.71
                tugger         15          5        0.6      0.908       0.88      0.623
               cabinet         15          6      0.909          1      0.995      0.819
                   str         15         10      0.912          1      0.995       0.78
                   box         15         10      0.575          1      0.888      0.635
              forklift         15          5      0.485          1      0.995      0.693
Speed: 0.2ms pre-process, 8.3ms inference, 13.9ms NMS per image at shape (32, 3, 640, 640)
```



```
Fusing layers...
YOLOv5s summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs
val: Scanning /content/yolov5_split/val/labels.cache... 15 images, 0 backgrounds, 0 corrupt: 100% 15/15 [00:00<?, ?it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:01<00:00,  1.56s/it]
                   all         15         36      0.984      0.993      0.995      0.862
                tugger         15          5      0.977          1      0.995      0.787
               cabinet         15          6      0.977          1      0.995      0.907
                   str         15         10      0.988          1      0.995       0.87
                   box         15         10          1      0.964      0.995      0.786
              forklift         15          5      0.981          1      0.995       0.96
Speed: 0.2ms pre-process, 8.8ms inference, 16.3ms NMS per image at shape (32, 3, 640, 640)
```

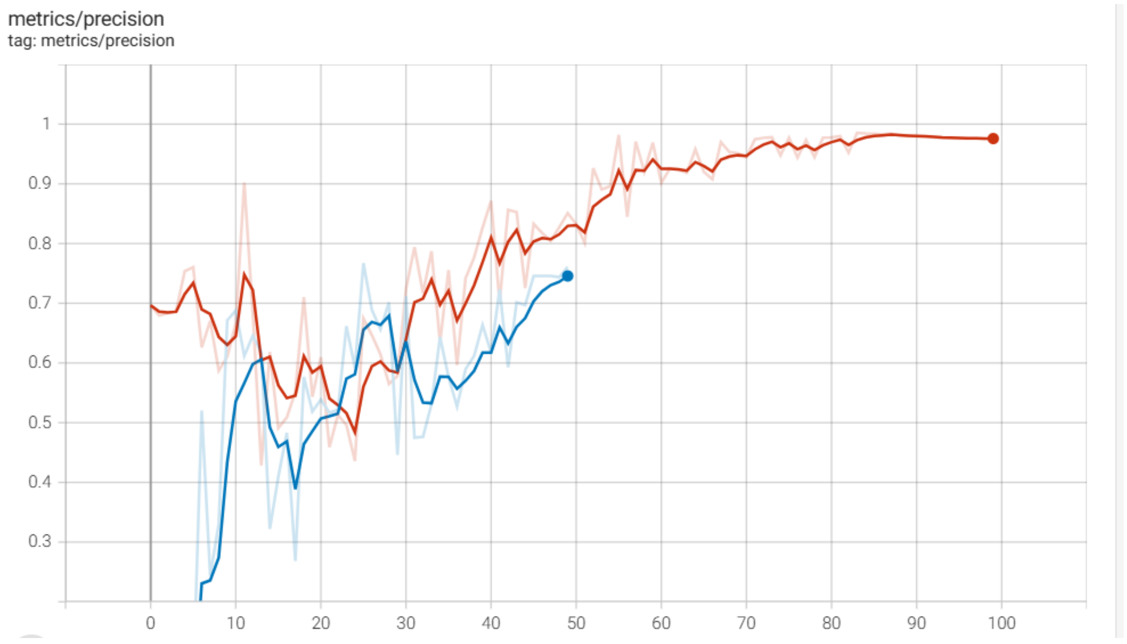
The per image inference speed of the first model is less than that of the second model, since the first model has less batch size than that of the first. So it takes less time to update the parameters.

- Tensorboard:

```
%load_ext tensorboard
%tensorboard --logdir runs/train
```

⇨ This shows the evolution of the models during **training as required.**

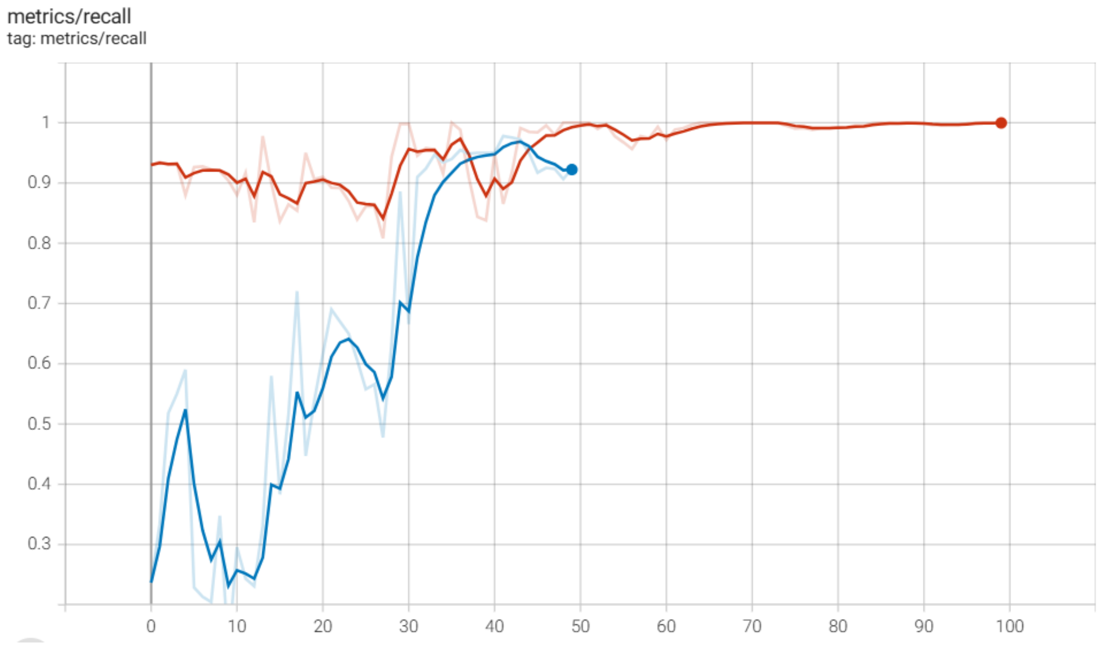
- Evolution of Precision:



The precision of the second model (tunned model – in red) is much higher than that of the first model (in blue) ensuring that **the second model predicts lower false positives.** This is logical since the second model is a tunned version of the first, and also, the second model has more space to learn from the data due to going over the data in 100 epochs > 50 epochs, and has higher batch size 12 > 8 to avoid possibility of over fitting.

An important observation is that the first model started the precision with less than 0.3, while the second model started it with almost 0.7. This is because the first model started from scratch, however, the second model is a fine tunned version tuning the first model.

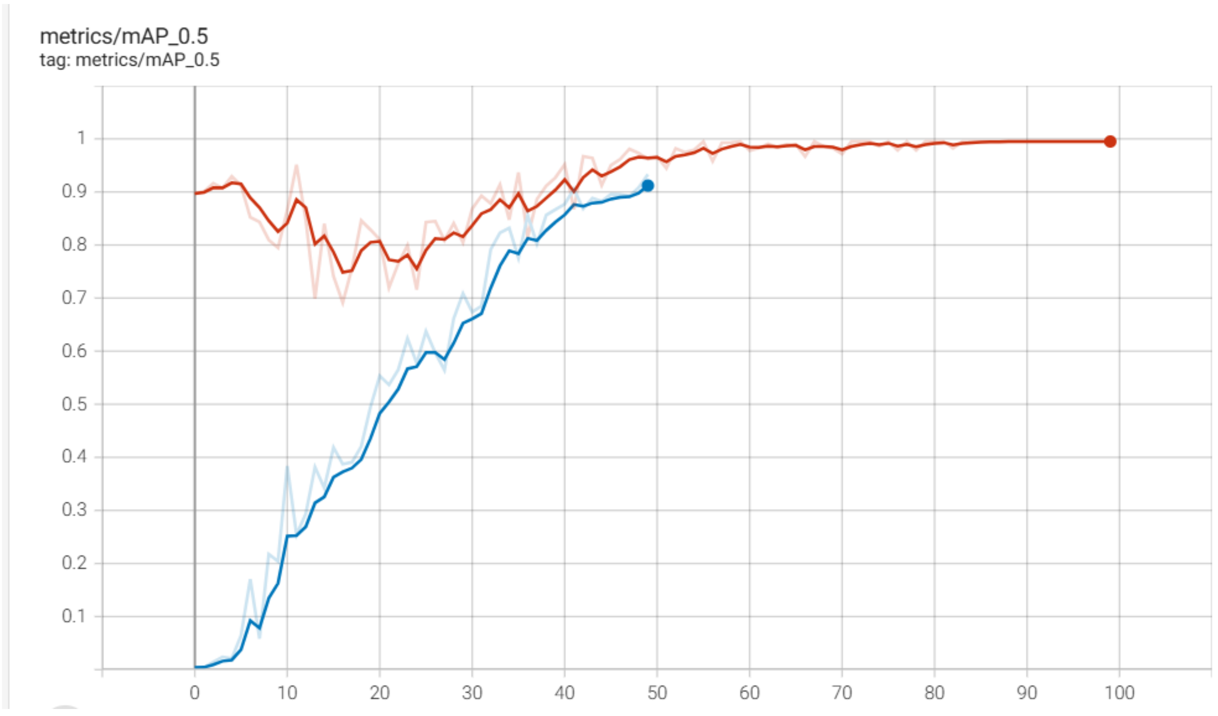
- Evolution of Recall:



The recall of the second model (in red) is also higher than that of the first model (in blue) ensuring that **the second model predicts lower false negatives.** This is logical since the second model is a tunned version of the first, and also, the second model has more space to learn from the data due to going over the data in 100 epochs > 50 epochs, and has higher batch size 12 > 8 to avoid possibility of over fitting.

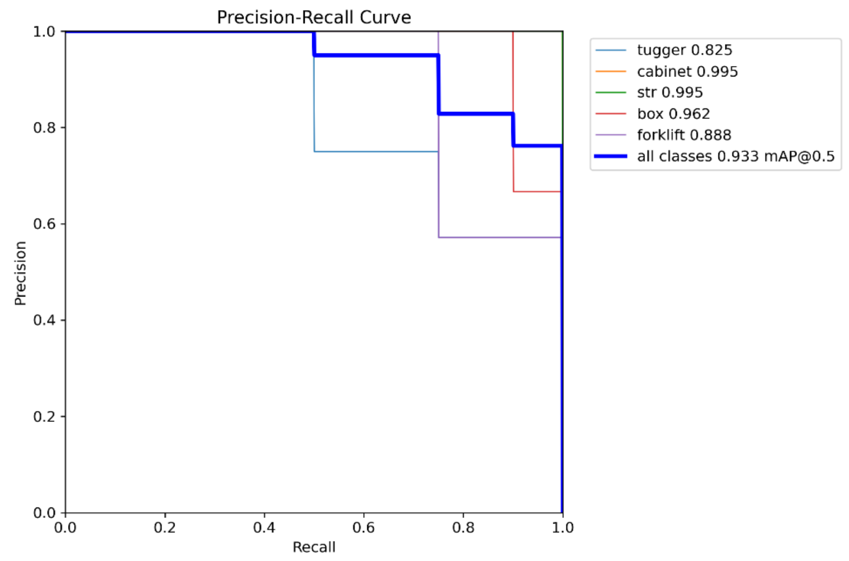
An important observation is that the first model started the recall with less than 0.3, while the second model started it with higher than 0.9. This is because the first model started from scratch, however, the second model is a fine tunned version tuning the first model.

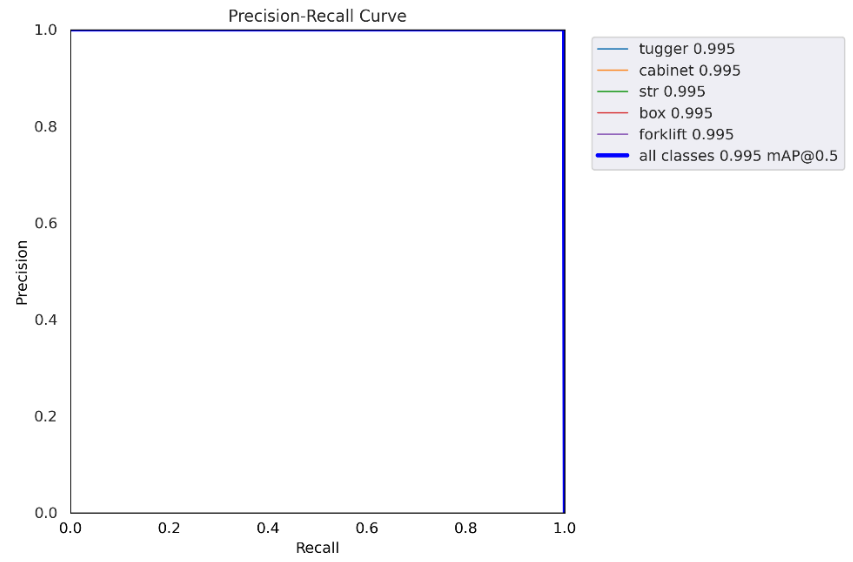
- Evolution of average precision with IoU = 0.5:



The second model is a fine tunned version of the first model. This is why it starts with high average precision. The final precision reached by the second model is very high, almost perfect. The average precision reached by the first model is almost 90%.

- ## Precision/Recall curve at IoU=0.5:

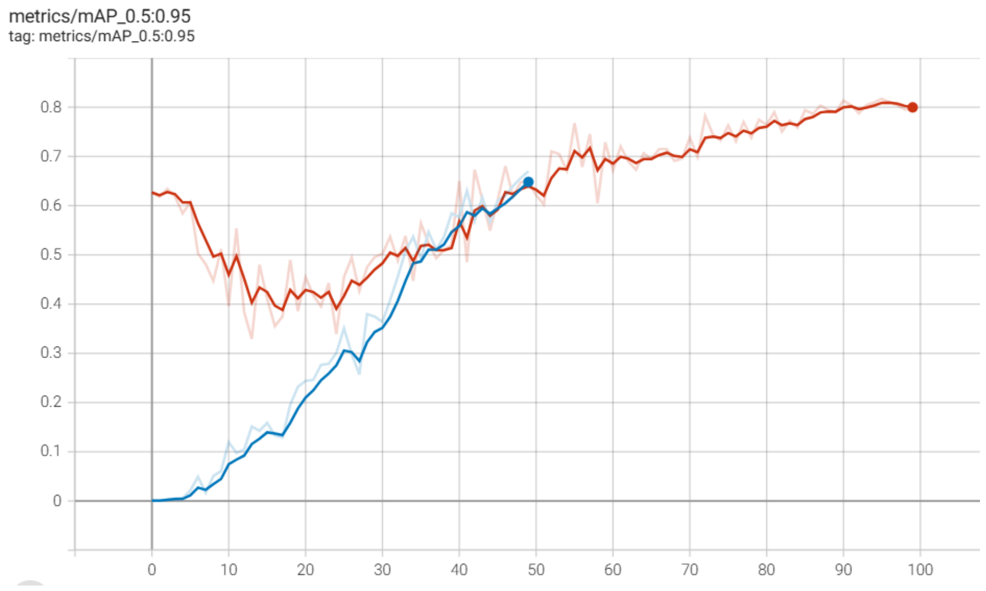




➔Both curves give us an information about the average precision at IoU=0.5. It can be seen that the second model (tunned model) results with higher average precision, and almost perfect scores in terms of precision and recall for all classes. Compared to the first run, the first run also provides almost perfect results for str, cabinet, and box. As for tugger and forklift they have lower precision which might be logical since in some cases they seem similar.

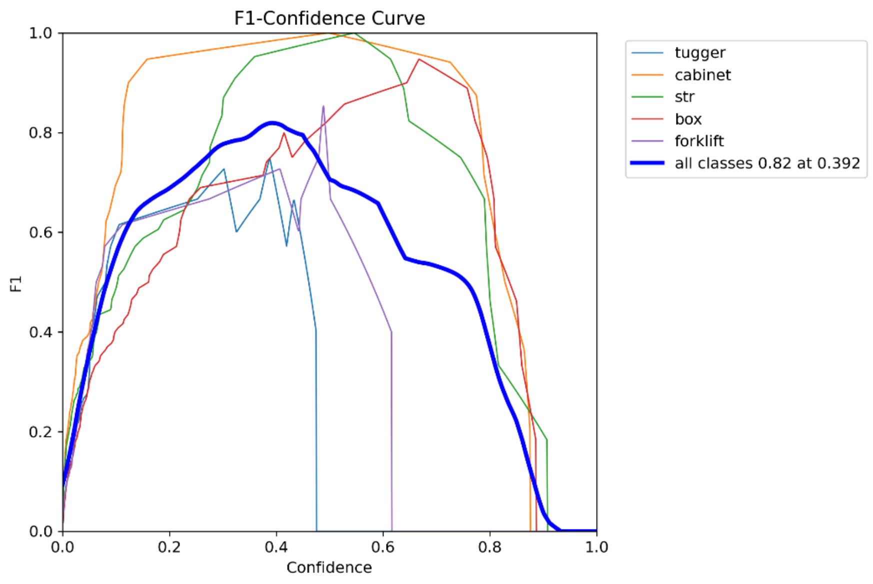
Generally, after changing the hyper parameters which is increasing number of epochs from 50 to 100, and batch size from 8 to 12, the model provided us better results in terms of average precision and recall.

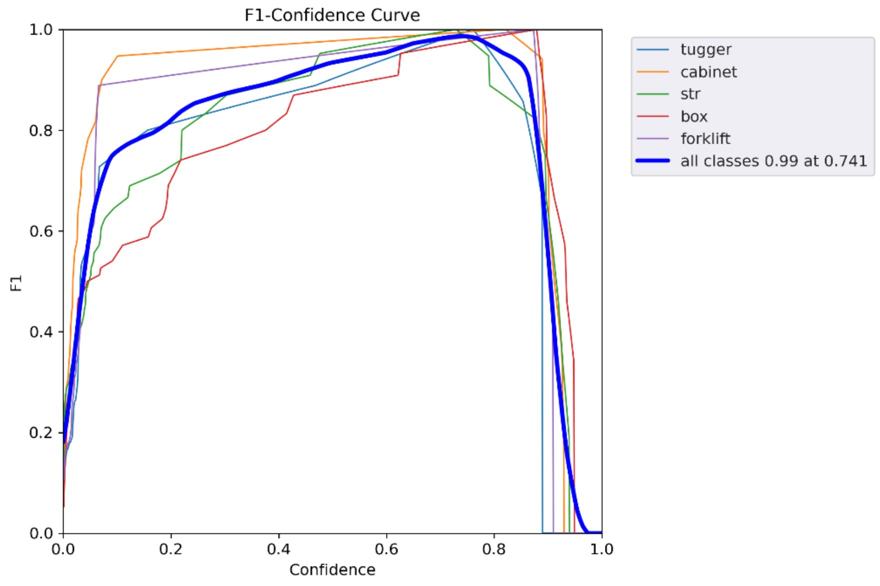
- Evolution of average precision with IoU = 0.95:



It can be seen that when the IoU was increased from 0.5 to 0.95, the average precision decreased. This indicates that the model is detecting an object but the object class is classified incorrectly. Also, like before, the second model which is a tunned version of the first model reaches higher average precision.

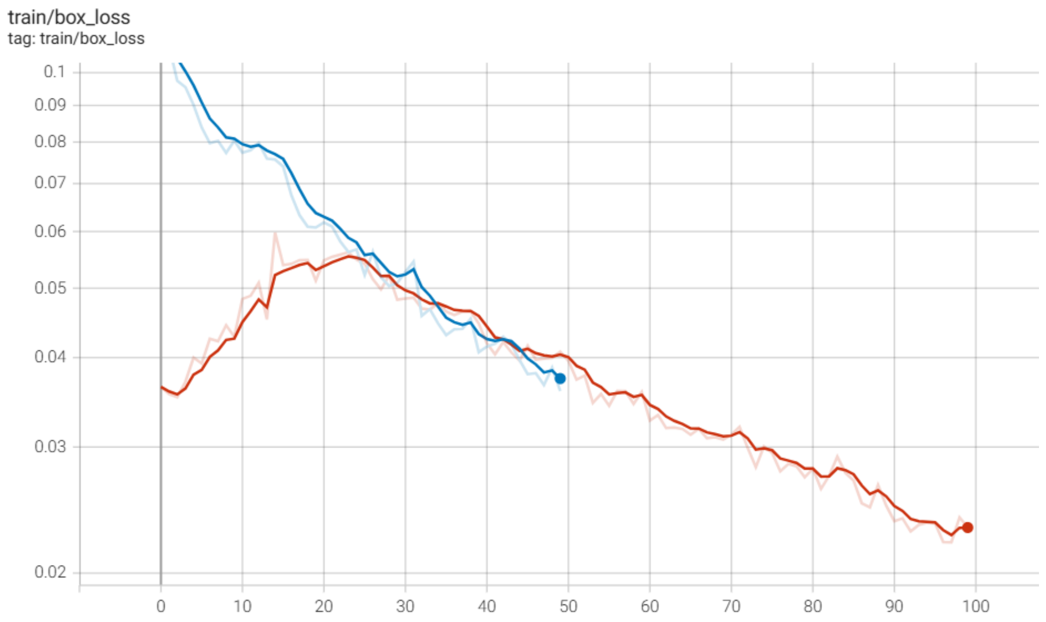
- ## F1 – Confidence Curve:

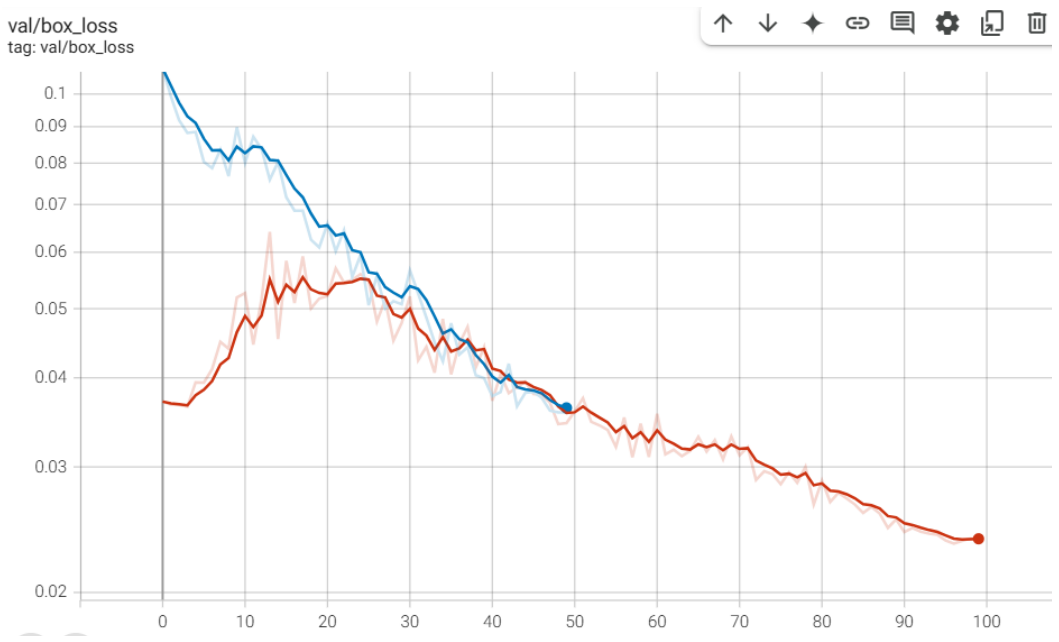




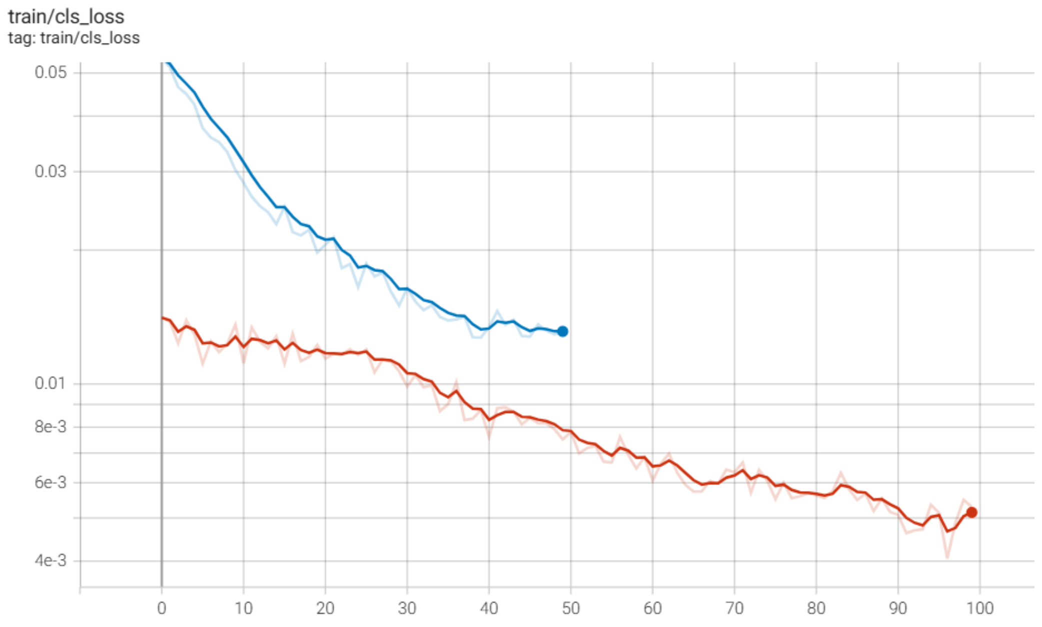
Comparing F1-confidence for both models, the first model has lower performance that peaks at almost 0.83 with confidence 0.392. This means that we should try to consider low confidence or else there will be drop in performance. As for the second model (tunned one), its f1 reaches 0.99 at confidence 0.741. The second model is clearly better than the first and recommends preserving high confidence bboxes for better results.

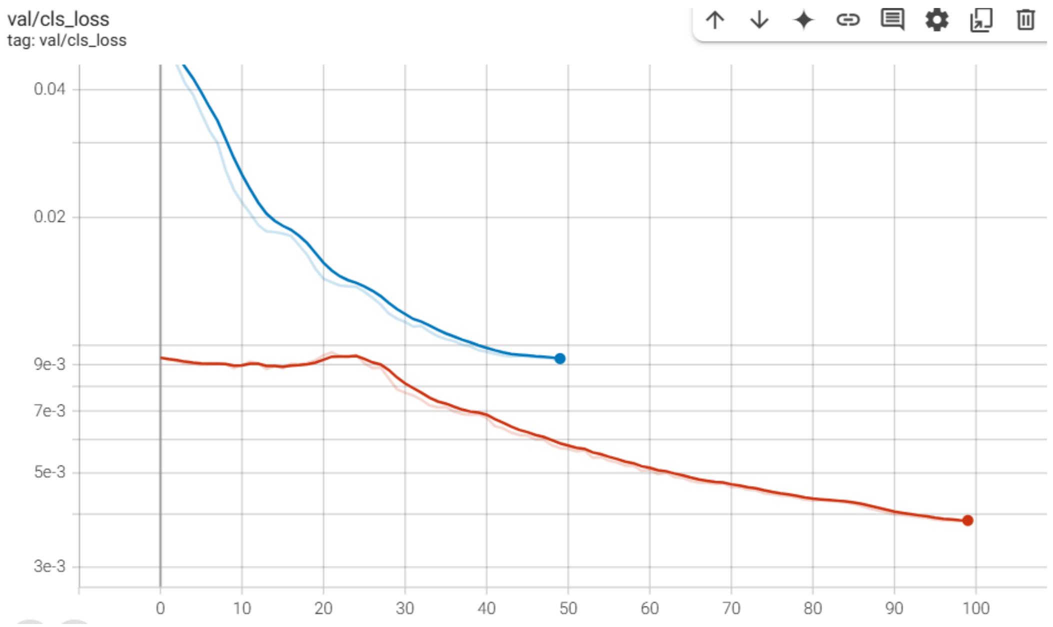
- Box Loss:


train/box_loss
tag: train/box_loss
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0
10
20
30
40
50
60
70
80
90
100

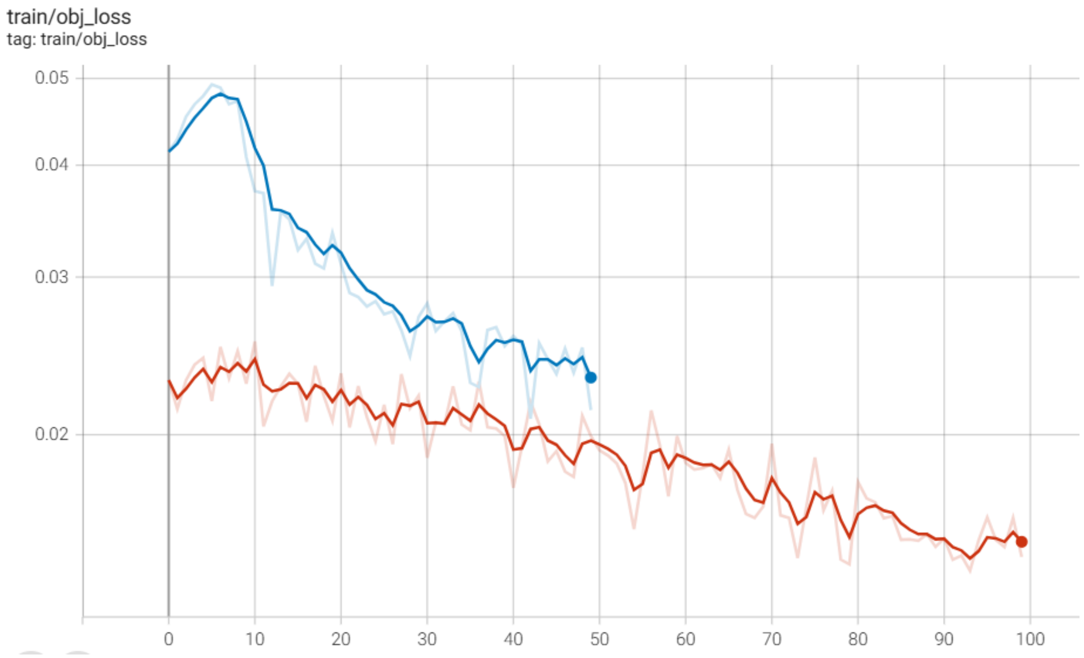

val/box_loss
tag: val/box_loss
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0
10
20
30
40
50
60
70
80
90
100

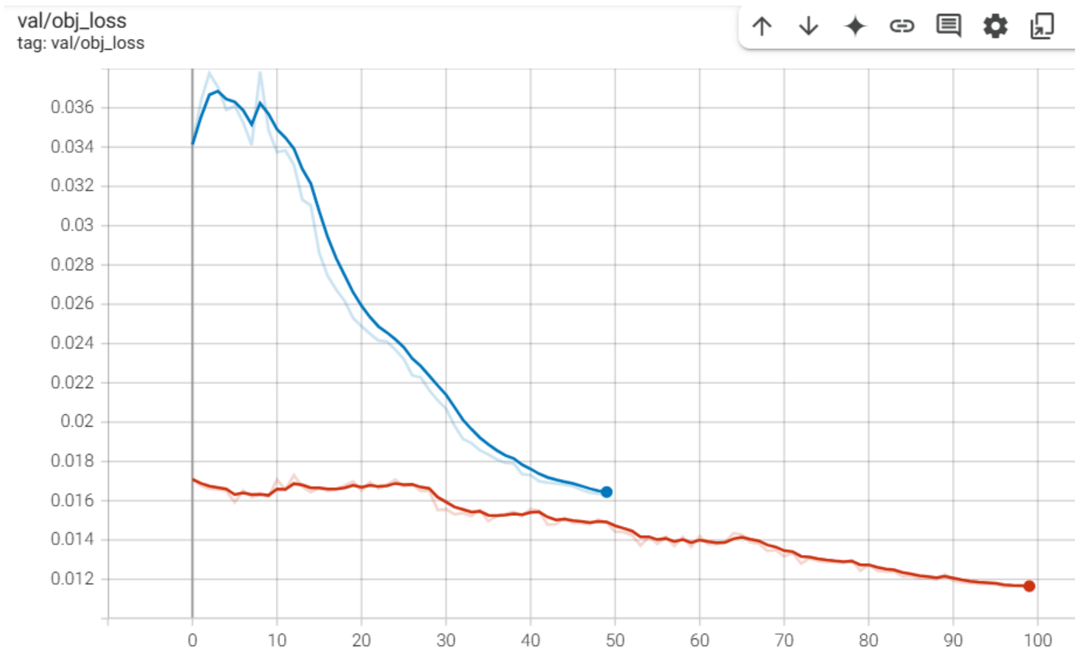
- Class Loss:


train/cls_loss
tag: train/cls_loss
0.05
0.03
0.01
8e-3
6e-3
4e-3
0
10
20
30
40
50
60
70
80
90
100


val/cls_loss
tag: val/cls_loss
0.04
0.02
9e-3
7e-3
5e-3
3e-3
0
10
20
30
40
50
60
70
80
90
100

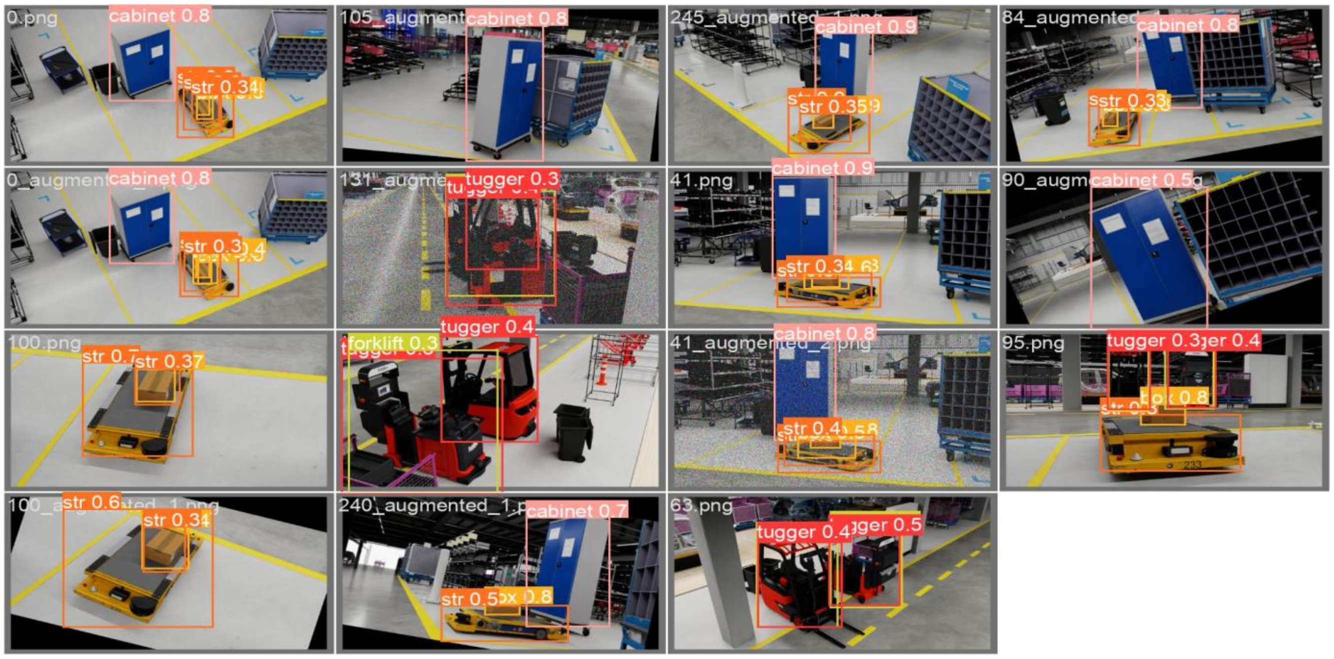
- Objectness Loss:

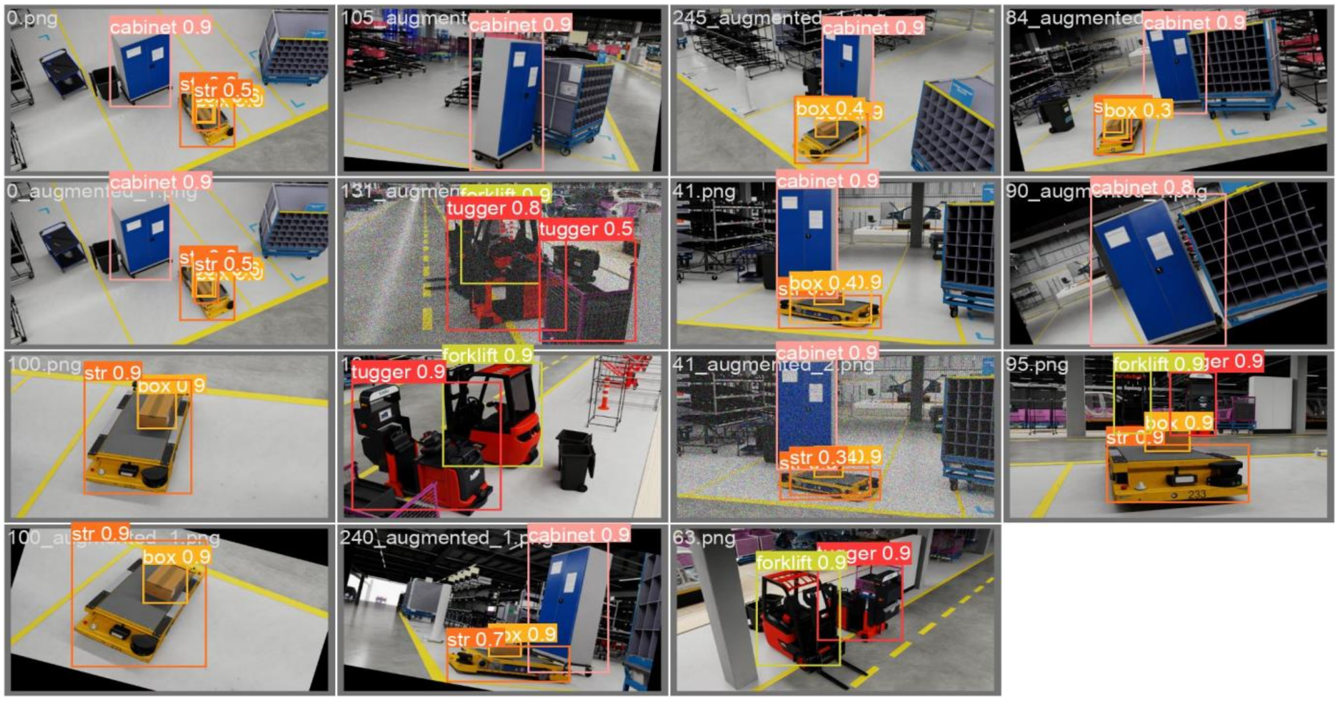




The losses are all decreasing as the model is trained or tested, this ensures that the model is not overfitting. Also, it is clear that the second model has lower losses than the first model which is logical since the second model which is tunned, makes less errors in predictions.

## - Detection Assessment:





Comparing the detection results of both models, the second model is more robust and is more confident when detecting. This is clear since less noisy bboxes are present in the predictions of the second model, also the second model achieves higher confidence scores.

## 7. *Calculating IoU:*

- Function to calculate IoU:

```python
# Set image dimensions as specified when training yolo
img_width = 640
img_height = 640

#convert yolo format to (x1, y1, x2, y2) that are not normalized
def yolo_to_xyxy(xc, yc, w, h):
    x1 = (xc - w / 2) * img_width
    y1 = (yc - h / 2) * img_height
    x2 = (xc + w / 2) * img_width
    y2 = (yc + h / 2) * img_height
    return [x1, y1, x2, y2]

#compute IoU between ground truth and predicted bbox
def compute_iou(boxA, boxB):
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])
    interArea = max(0, xB - xA) * max(0, yB - yA) #intersection of the areas
    boxAArea = (boxA[2] - boxA[0]) * (boxA[3] - boxA[1]) #area of boxA
    boxBArea = (boxB[2] - boxB[0]) * (boxB[3] - boxB[1]) #area of boxB
    iou = interArea / (boxAArea + boxBArea - interArea) #getting intersection over union
    return iou
```

- Function to load the labels:

```python
#load predicted label files by yolo into a dictionary
def load_predictions(pred_dir):
    pred_dict = {}
    for file in os.listdir(pred_dir):
        if file.endswith(".txt"):
            file_path = os.path.join(pred_dir, file)
            with open(file_path, "r") as f:
                lines = f.readlines()
            boxes = []
            for line in lines:
                parts = list(map(float, line.strip().split()))
                if len(parts) == 6:
                    parts = parts[:5]  #retreive only class id and coordinates from yolo file
                boxes.append(parts)  # [class_id, xc, yc, w, h]
            pred_dict[file] = boxes
    return pred_dict
```

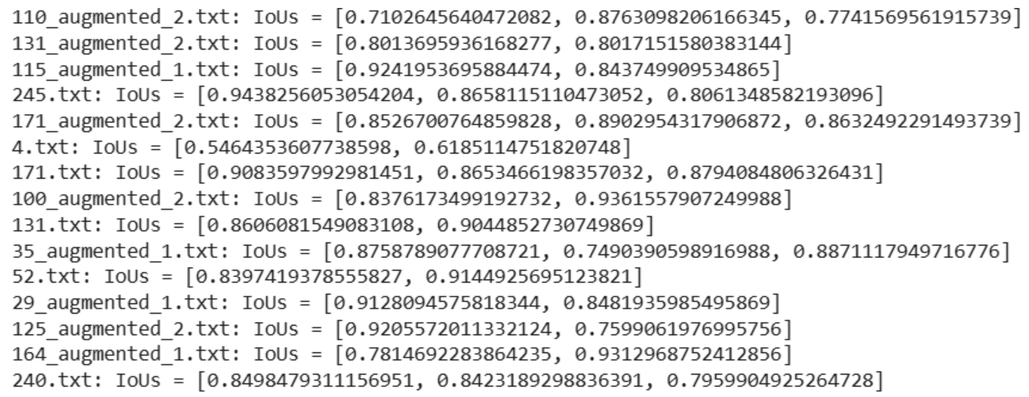
- Function to calculate all the IoUs:

```python
from glob import glob

def compute_all_ious(gt_dir, pred_dir):
    pred_dict = load_predictions(pred_dir) #path holding predicted bboxes
    results = {}

    for label_file in glob(os.path.join(gt_dir, "*.txt")):
        filename = os.path.basename(label_file)
        with open(label_file, "r") as f:
            gt_lines = f.readlines()

        gt_boxes = []
        for line in gt_lines:
            parts = list(map(float, line.strip().split()))
            _, xc, yc, w, h = parts
            gt_boxes.append(yolo_to_xyxy(xc, yc, w, h))

        pred_boxes = []
        if filename in pred_dict:
            for pred in pred_dict[filename]:
                _, xc, yc, w, h = pred
                pred_boxes.append(yolo_to_xyxy(xc, yc, w, h))

        image_ious = []
        for gt_box in gt_boxes:
            best_iou = 0
            for pred_box in pred_boxes:
                iou = compute_iou(gt_box, pred_box)
                if iou > best_iou:
                    best_iou = iou  #preserve only the bboxes with best iou
            image_ious.append(best_iou)

        results[filename] = image_ious

    return results
```

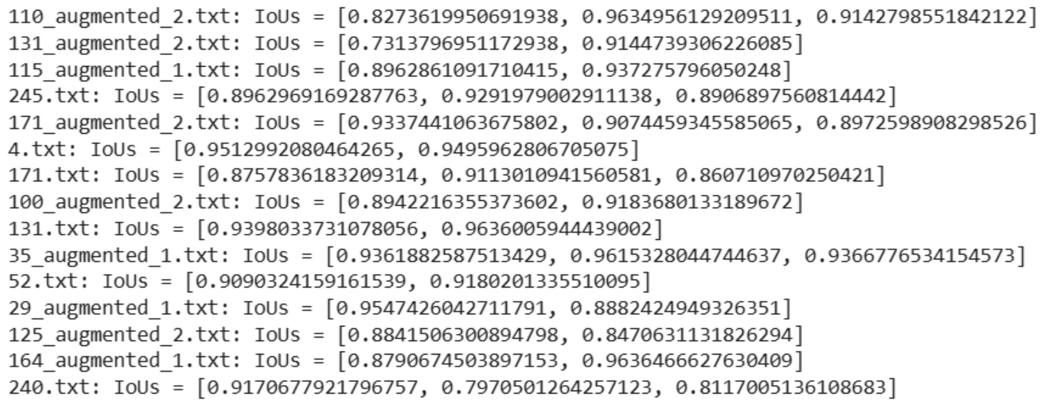
- Calculating the IoUs using both models:

```python
# Run the function on your data
gt_dir = '../yolov5_split/val/labels'
first_model_labels = 'runs/detect/exp5/labels'  #path holding the labels of test data by first model
second_model_labels = 'runs/detect/exp4/labels'  #path holding the labels of test data by second model
ious1 = compute_all_ious(gt_dir, first_model_labels)
ious2 = compute_all_ious(gt_dir, second_model_labels)

print(ious1.items())

# Print results
print('IoUs for the 1st model are as follows: ')
for img_name, iou_list in ious1.items():
    print(f"{img_name}: IoUs = {iou_list}")

print('IoUs for the 2nd model are as follows: ')
for img_name, iou_list in ious2.items():
    print(f"{img_name}: IoUs = {iou_list}")
```

- IoU Evaluation:



```
110_augmented_2.txt: IoUs = [0.7102645640472082, 0.8763098206166345, 0.7741569561915739]
131_augmented_2.txt: IoUs = [0.8013695936168277, 0.8017151580383144]
115_augmented_1.txt: IoUs = [0.9241953695884474, 0.843749909534865]
245.txt: IoUs = [0.9438256053054204, 0.8658115110473052, 0.8061348582193096]
171_augmented_2.txt: IoUs = [0.8526700764859828, 0.8902954317906872, 0.8632492291493739]
4.txt: IoUs = [0.5464353607738598, 0.6185114751820748]
171.txt: IoUs = [0.9083597992981451, 0.8653466198357032, 0.8794084806326431]
100_augmented_2.txt: IoUs = [0.8376173499192732, 0.9361557907249988]
131.txt: IoUs = [0.8606081549083108, 0.9044852730749869]
35_augmented_1.txt: IoUs = [0.8758789077708721, 0.7490390598916988, 0.8871117949716776]
52.txt: IoUs = [0.8397419378555827, 0.9144925695123821]
29_augmented_1.txt: IoUs = [0.9128094575818344, 0.8481935985495869]
125_augmented_2.txt: IoUs = [0.9205572011332124, 0.7599061976995756]
164_augmented_1.txt: IoUs = [0.7814692283864235, 0.9312968752412856]
240.txt: IoUs = [0.8498479311156951, 0.8423189298836391, 0.7959904925264728]
```



```
110_augmented_2.txt: IoUs = [0.8273619950691938, 0.9634956129209511, 0.9142798551842122]
131_augmented_2.txt: IoUs = [0.7313796951172938, 0.9144739306226085]
115_augmented_1.txt: IoUs = [0.8962861091710415, 0.937275796050248]
245.txt: IoUs = [0.8962969169287763, 0.9291979002911138, 0.8906897560814442]
171_augmented_2.txt: IoUs = [0.9337441063675802, 0.9074459345585065, 0.8972598908298526]
4.txt: IoUs = [0.9512992080464265, 0.9495962806705075]
171.txt: IoUs = [0.8757836183209314, 0.9113010941560581, 0.860710970250421]
100_augmented_2.txt: IoUs = [0.8942216355373602, 0.9183680133189672]
131.txt: IoUs = [0.9398033731078056, 0.9636005944439002]
35_augmented_1.txt: IoUs = [0.9361882587513429, 0.9615328044744637, 0.9366776534154573]
52.txt: IoUs = [0.9090324159161539, 0.9180201335510095]
29_augmented_1.txt: IoUs = [0.9547426042711791, 0.8882424949326351]
125_augmented_2.txt: IoUs = [0.8841506300894798, 0.8470631131826294]
164_augmented_1.txt: IoUs = [0.8790674503897153, 0.9636466627630409]
240.txt: IoUs = [0.9170677921796757, 0.7970501264257123, 0.8117005136108683]
```

The IoU of the second model is in general higher than that of the first model which is logical since the second model is a fine tunned version of the first, and the second model relies on 100 epochs to learn from data.

# Model Deployment & Inference:

## 1. Export to ONNX:

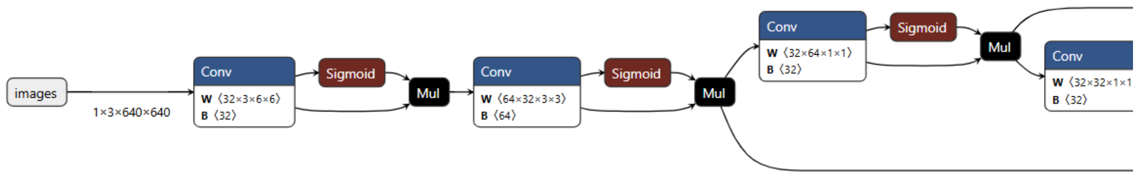
```
%cd /content/yolov5/
```

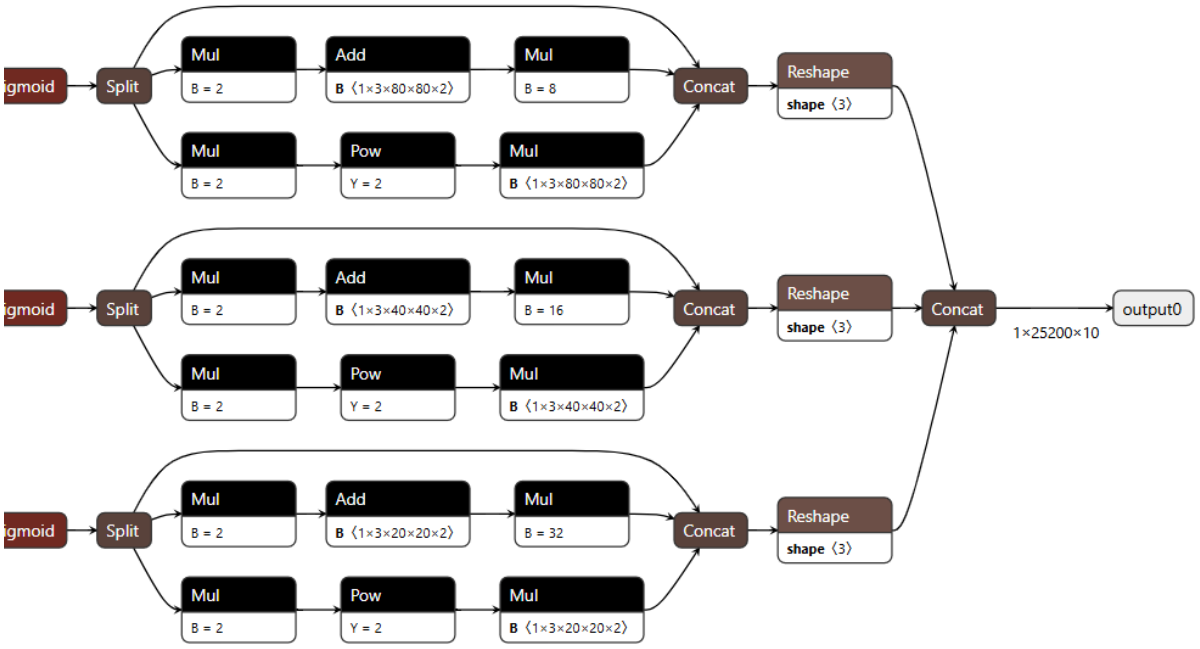
```
from yolov5 import export
from google.colab import files
import shutil

export.run(
    weights='runs/train/tunned_run/weights/best.pt',  #path to .pt parameters (best weights attained)
    imgsz=[640, 640], #image size
    include=['onnx'] #export format type
)

# Rename tuned ONNX model to second_onnx.onnx
shutil.move('runs/train/tunned_run/weights/best.onnx', 'second_onnx.onnx')

export.run(
    weights='runs/train/first_run2/weights/best.pt',  #path to .pt parameters (best weights attained)
    imgsz=[640, 640], #image size
    include=['onnx'] #export format type
)

# Rename untuned ONNX model to first_onnx.onnx
shutil.move('runs/train/first_run2/weights/best.onnx', 'first_onnx.onnx')

#download the onnx file
files.download('first_onnx.onnx')
files.download('second_onnx.onnx')
```

## 2. Inference Example with ONNX:

```
!python detect.py --weights runs/train/tunned_run/weights/best.onnx --source /content/yolov5_split/val/images --img 640
```

## 3. Netron:





The model is made of a very big neural network with many layers, multiple activation functions etc… The input is (1 x 3 x 640) which is logical since RGB image and of size 640. As for the output it is (1 x 25200 x 10) which reflects that we detect 25200 total bounding boxes for 1 image, for each bounding box there are 10 parameters as follows:

[ x, y, w, h, objectness, class1, class2, class3, class4, class5]

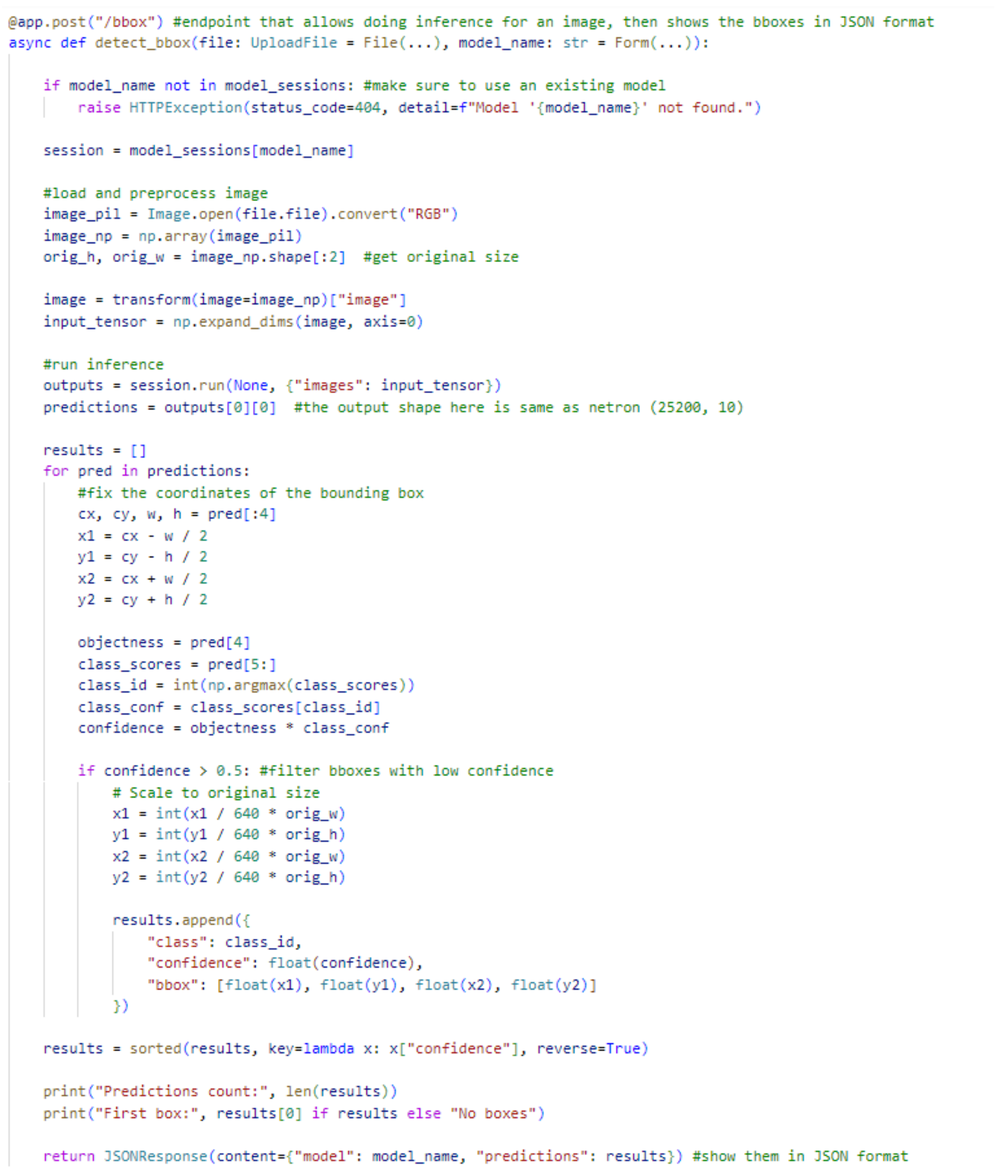
## 4. API:

### - Model Listing:

```
app = FastAPI()

#load all ONNX models from the directory they are saved at
model_directory = "C:\\Users\\PC\\OneDrive - Lebanese American University\\inmind\\ML track\\final project\\submission\\Model Deployment & Inference\\Netron"
model_sessions = {
    model_name: ort.InferenceSession(os.path.join(model_directory, model_name), providers=["CPUExecutionProvider"]) #specify using CPU
    for model_name in os.listdir(model_directory) if model_name.endswith(".onnx")
}
```

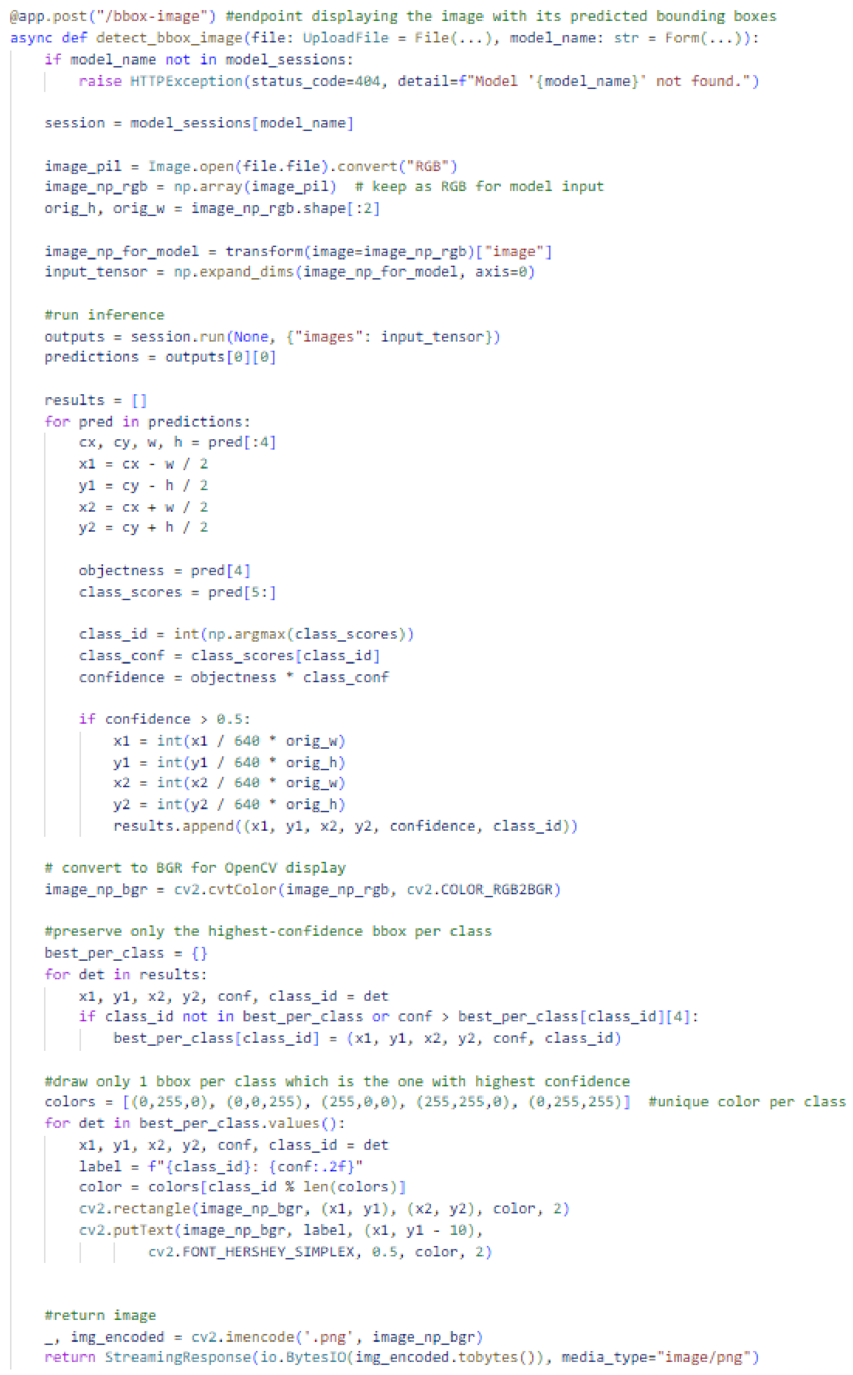
```
@app.get("/models") #endpoint holding the names of trained yolo models
def list_models():
    return {"models": list(model_sessions.keys())}
```

- ## Inference / BBoxes:



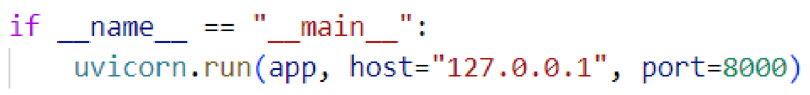
```python
@app.post("/bbox") #endpoint that allows doing inference for an image, then shows the bboxes in JSON format
async def detect_bbox(file: UploadFile = File(...), model_name: str = Form(...)):

    if model_name not in model_sessions: #make sure to use an existing model
        raise HTTPException(status_code=404, detail=f"Model '{model_name}' not found.")

    session = model_sessions[model_name]

    #load and preprocess image
    image_pil = Image.open(file.file).convert("RGB")
    image_np = np.array(image_pil)
    orig_h, orig_w = image_np.shape[:2]  #get original size

    image = transform(image=image_np)["image"]
    input_tensor = np.expand_dims(image, axis=0)

    #run inference
    outputs = session.run(None, {"images": input_tensor})
    predictions = outputs[0][0]  #the output shape here is same as netron (25200, 10)

    results = []
    for pred in predictions:
        #fix the coordinates of the bounding box
        cx, cy, w, h = pred[:4]
        x1 = cx - w / 2
        y1 = cy - h / 2
        x2 = cx + w / 2
        y2 = cy + h / 2

        objectness = pred[4]
        class_scores = pred[5:]
        class_id = int(np.argmax(class_scores))
        class_conf = class_scores[class_id]
        confidence = objectness * class_conf

        if confidence > 0.5: #filter bboxes with low confidence
            # Scale to original size
            x1 = int(x1 / 640 * orig_w)
            y1 = int(y1 / 640 * orig_h)
            x2 = int(x2 / 640 * orig_w)
            y2 = int(y2 / 640 * orig_h)

            results.append({
                "class": class_id,
                "confidence": float(confidence),
                "bbox": [float(x1), float(y1), float(x2), float(y2)]
            })

    results = sorted(results, key=lambda x: x["confidence"], reverse=True)

    print("Predictions count:", len(results))
    print("First box:", results[0] if results else "No boxes")

    return JSONResponse(content={"model": model_name, "predictions": results}) #show them in JSON format
```

- ## Inference Image with BBoxes:

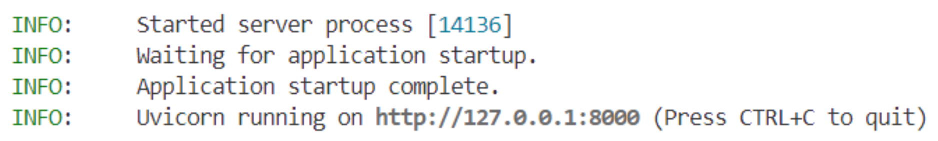


```python
@app.post("/bbox-image") #endpoint displaying the image with its predicted bounding boxes
async def detect_bbox_image(file: UploadFile = File(...), model_name: str = Form(...)):
    if model_name not in model_sessions:
        raise HTTPException(status_code=404, detail=f"Model '{model_name}' not found.")

    session = model_sessions[model_name]

    image_pil = Image.open(file.file).convert("RGB")
    image_np_rgb = np.array(image_pil)  # keep as RGB for model input
    orig_h, orig_w = image_np_rgb.shape[:2]

    image_np_for_model = transform(image=image_np_rgb)["image"]
    input_tensor = np.expand_dims(image_np_for_model, axis=0)

    #run inference
    outputs = session.run(None, {"images": input_tensor})
    predictions = outputs[0][0]

    results = []
    for pred in predictions:
        cx, cy, w, h = pred[:4]
        x1 = cx - w / 2
        y1 = cy - h / 2
        x2 = cx + w / 2
        y2 = cy + h / 2

        objectness = pred[4]
        class_scores = pred[5:]

        class_id = int(np.argmax(class_scores))
        class_conf = class_scores[class_id]
        confidence = objectness * class_conf

        if confidence > 0.5:
            x1 = int(x1 / 640 * orig_w)
            y1 = int(y1 / 640 * orig_h)
            x2 = int(x2 / 640 * orig_w)
            y2 = int(y2 / 640 * orig_h)
            results.append((x1, y1, x2, y2, confidence, class_id))

    # convert to BGR for OpenCV display
    image_np_bgr = cv2.cvtColor(image_np_rgb, cv2.COLOR_RGB2BGR)

    #preserve only the highest-confidence bbox per class
    best_per_class = {}
    for det in results:
        x1, y1, x2, y2, conf, class_id = det
        if class_id not in best_per_class or conf > best_per_class[class_id][4]:
            best_per_class[class_id] = (x1, y1, x2, y2, conf, class_id)

    #draw only 1 bbox per class which is the one with highest confidence
    colors = [(0,255,0), (0,0,255), (255,0,0), (255,255,0), (0,255,255)]  #unique color per class
    for det in best_per_class.values():
        x1, y1, x2, y2, conf, class_id = det
        label = f"{class_id}: {conf:.2f}"
        color = colors[class_id % len(colors)]
        cv2.rectangle(image_np_bgr, (x1, y1), (x2, y2), color, 2)
        cv2.putText(image_np_bgr, label, (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)


    #return image
    _, img_encoded = cv2.imencode('.png', image_np_bgr)
    return StreamingResponse(io.BytesIO(img_encoded.tobytes()), media_type="image/png")
```

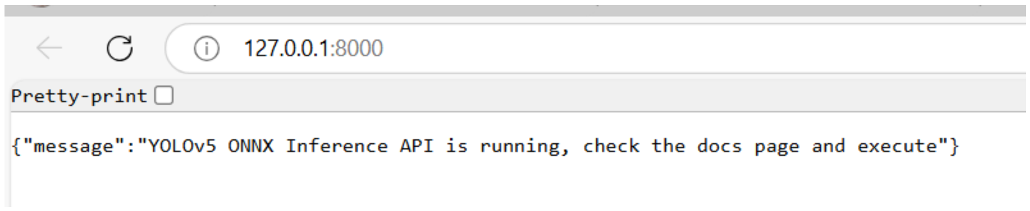
- ## Launching the API:



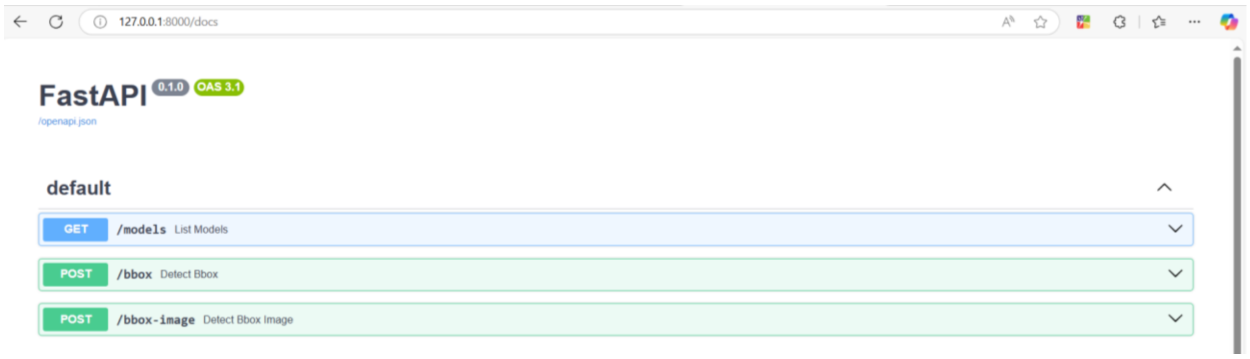
```python
if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

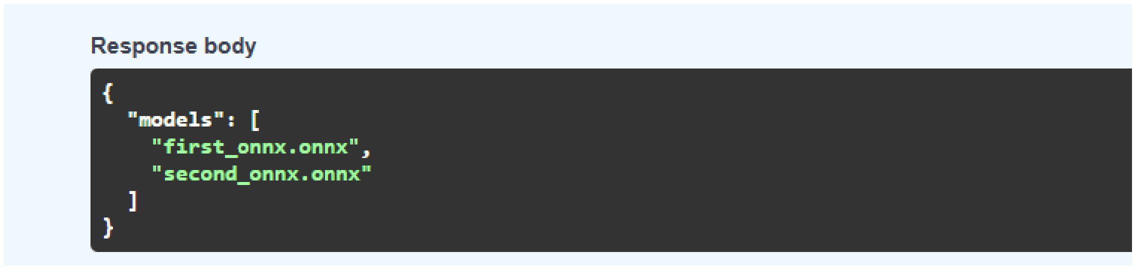


```
INFO:     Started server process [14136]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

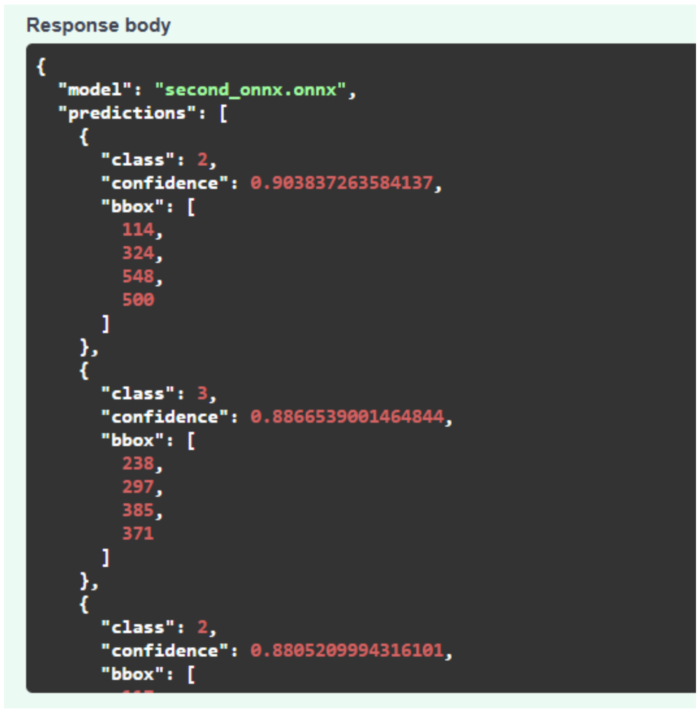


```
{"message":"YOLOv5 ONNX Inference API is running, check the docs page and execute"}
```



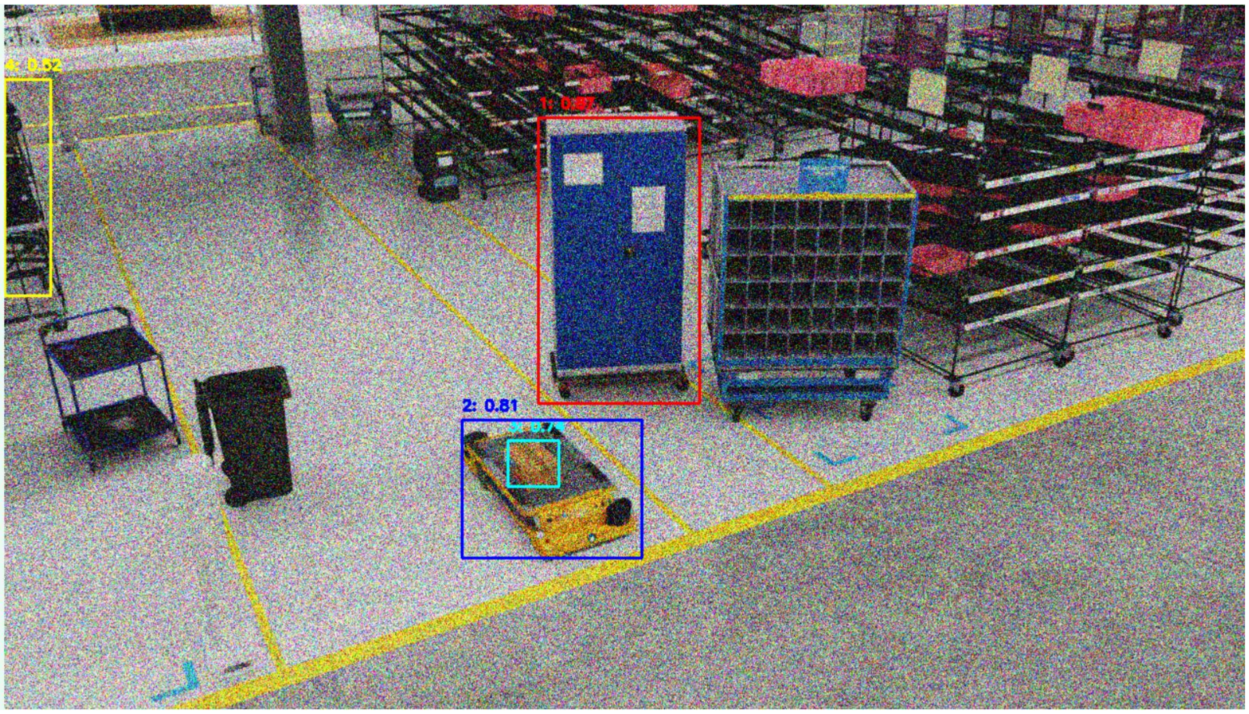

Response body

```
{
  "models": [
    "first_onnx.onnx",
    "second_onnx.onnx"
  ]
}
```



Response body

```
{
  "model": "second_onnx.onnx",
  "predictions": [
    {
      "class": 2,
      "confidence": 0.903837263584137,
      "bbox": [
        114,
        324,
        548,
        500
      ]
    },
    {
      "class": 3,
      "confidence": 0.8866539001464844,
      "bbox": [
        238,
        297,
        385,
        371
      ]
    },
    {
      "class": 2,
      "confidence": 0.8805209994316101,
      "bbox": [
```



## 5. *Docker:*

- Requirements:

```
fastapi
uvicorn[standard]
onnxruntime
albumentations
opencv-python-headless
pillow
numpy
scikit-learn
torch
```

- Dockerfile:

```
FROM python:3.10-slim

# Create and set working directory
RUN mkdir /app
WORKDIR /app

# Install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy all files into the container
COPY . .

# Expose port
EXPOSE 8000

# Run FastAPI with uvicorn
CMD ["uvicorn", "inference_api:app", "--host", "0.0.0.0", "--port", "8000"]
```

- Steps:

```
1. Go to the directory where the dockerfile, requirements.txt, and inference_api.py are all put
2. Run the following command on the terminal ==> docker build -t yolo-api .
3. Run the following command on the terminal ==> docker run -p 8000:8000 yolo-api
4. Insert the following link on the browser: http://localhost:8000/docs
5. You are ready to use the API
6. Once used it you can stop the run either using " Ctrl + C" on the terminal, or ==> docker stop 'container ID'
(The container ID is retrieved from docker ps)
```