# Real Time Object Detection with Yolov5

ENEL 645 Project Report

*Submitted by*

Group No: 9

**Deep Vyas 30139014**

**Mit Patel 30141193**

Dataset**: VOC 2007** (The PASCAL Visual Object Classes Challenge 2007)

Algorithm**: YOLO v5**

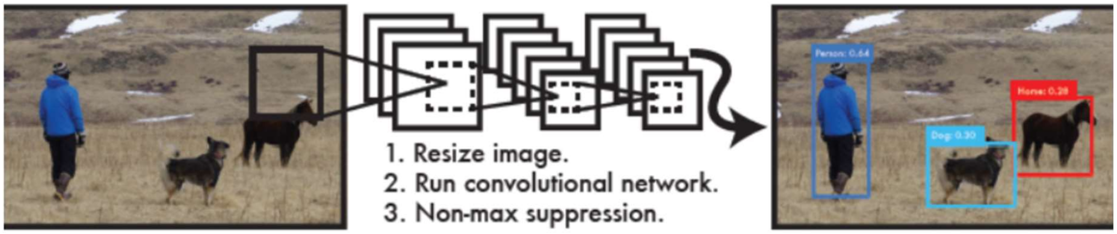
Project Link: https://github.com/vyasdeep668/Yolo-Object-Detection

# Table of Contents

# 1. Project Overview

## 1.1 Introduction

YOLO stands for you look only once. In YOLO algorithm the object detection problem is reframed as a single regression problem gives bounding box coordinates and class probabilities corresponding to the input image pixels.

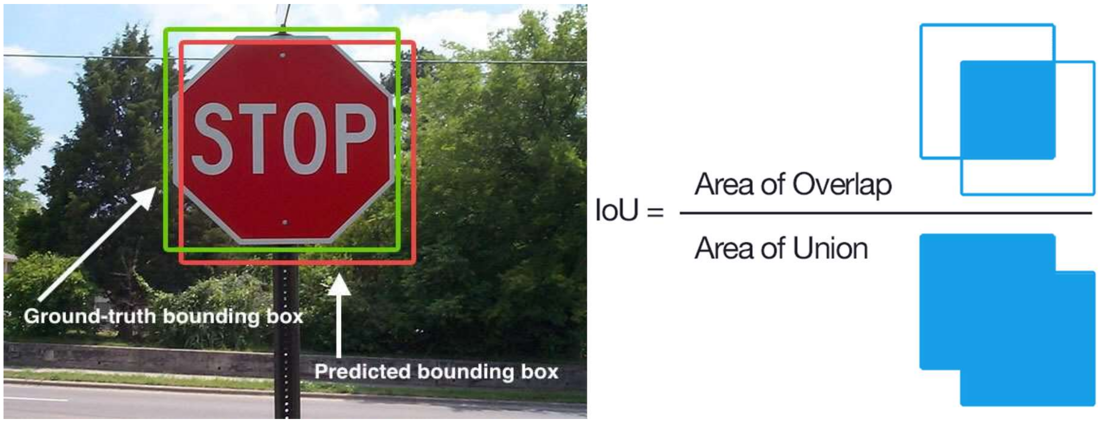


YOLO is a state-of-the-art algorithm. There are various other object detection approaches present in the market which must be discussed to get an idea of speed and accuracy of YOLO algorithm. One of the approaches in object detection is the Deformable Parts Model (DPM) which uses a sliding window approach in which a classifier, for a particular object is ran at evenly spaced over the entire image. This technique can take a long time and significant storage capacity to process data because most of the portion of the image which does not contain the target object is also iterated. Other recent development in the field of object detection is Region-base Convolutional Neural Network (R-CNN). It uses segmentation to generate potential bounding boxes around the objects and then a classifier is running on these proposed boxes. After classification, the bounding boxes are refined in post processing stage to in which duplicate detections are eliminated and the boxes are rescored based on other objects in the scene. These complicated pipelines are difficult to optimize because each individual component must be trained separately.

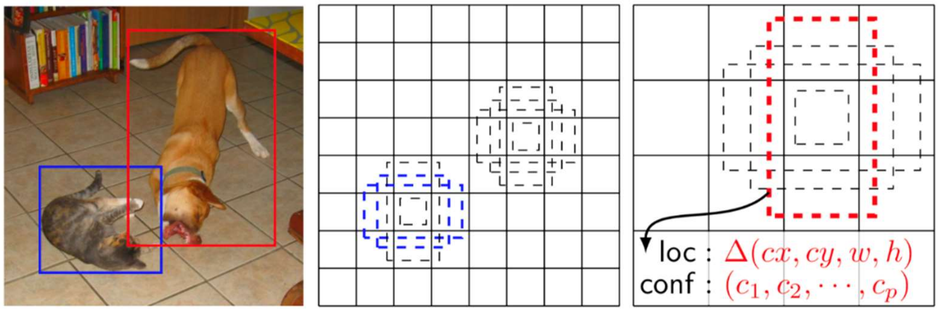
As compared to the methods discussed in the previous paragraph, in YOLO algorithm separate components of object detection are unified a single neural network. In YOLO design end to end training and real time speeds are enabled along with maintain high average precision. The above features are achieved because YOLO reasons globally about the entire image and all the objects in the image. Bounding Boxes across all the target classes are simultaneously predicted by it.

# 2. Object Detection

In YOLO algorithm, entire image is divided into a S*S grid. The grid cell in which the object center is located is responsible for the detection of that object. For multiple object detection within a grid cell the concept of anchor boxes is used. Before discussing anchor boxes an important metric which is Intersection Over Union (IOU) is defined in the following image.



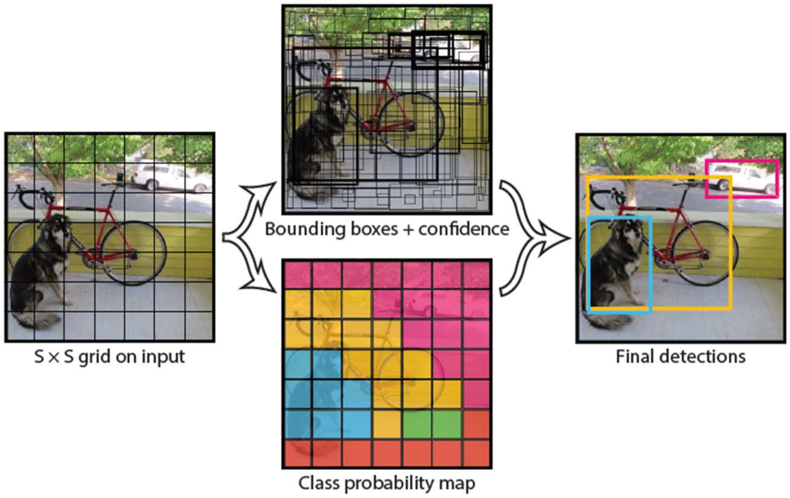
Anchor boxes in a particular grid cell which intersects with the bounding boxes of the object and who's the intersection whose IOU is greater than 50% is responsible for detection of that object. Thus, for multiple object detection within an image the pair of, grid cell containing the object midpoint and the anchor box whose IOU is greater than 50% for that object, is responsible for object detection. The concept of anchor box for objection is depicted below in the image.

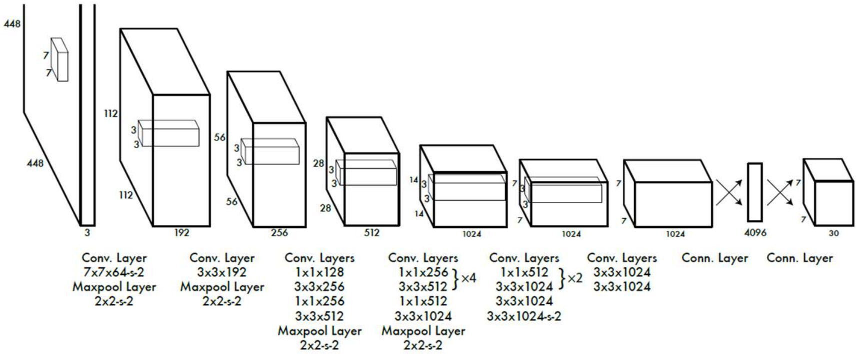

loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$

(a) Image with GT boxes (b) 8 × 8 feature map (c) 4 × 4 feature map

In YOLO algorithm, each of the S*S grid cell predicts B anchor boxes. The output of these grid cells is B*(C+5) tensor which consists of the following: x, y, w, h, confidence scores and the C class probabilities for each of the B anchor boxes. (x,

y) is the coordinate of the center of the bounding box, (h, w) represents height and width of the bounding boxes. The confidence scores represent how confident the model is that the box contains an object and is given by Pr($Object$)*IOU. The C conditional class probabilities are given by Pr($Class_i|Object$).Thus, the final output is a tensor of size S*S*B*(5+C). The image given below represents object detection using YOLO algorithm.





In the above diagram the neural network structure of the model consists of 24 convolutional layers and 2 fully connected layers. The convolutional layers extract all the features from the image. They convert the 2D input data into 1D data such that the spatial information of the data is preserved. This 1D data is then fed to the fully connected neural network which predicts the output probabilities and coordinates.

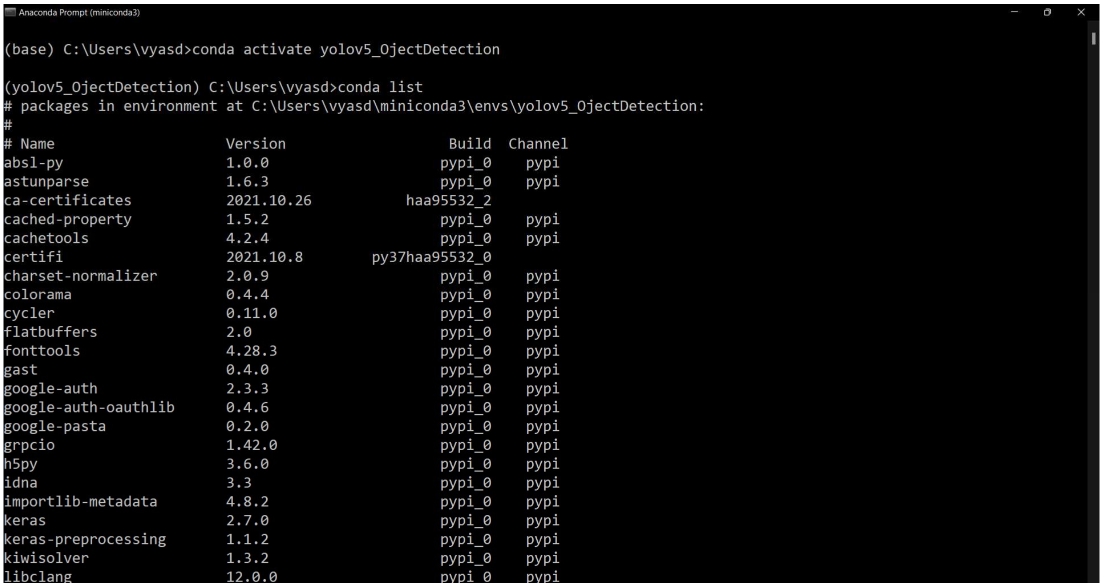
# 3. Project Design

## 3.1 Environment Setup

First, we need to create separate conda environment with python 3.7 to install all project dependencies on it.  Using below command we can create new conda environment.

*Conda create -n yolov5_ObjectDetection python=3.7*

Below image shows Conda Environment for our object detection project.



```
(base) C:\Users\vyasd>conda activate yolov5_OjectDetection

(yolov5_OjectDetection) C:\Users\vyasd>conda list
# packages in environment at C:\Users\vyasd\miniconda3\envs\yolov5_OjectDetection:
#
# Name                    Version                   Build  Channel
absl-py                   1.0.0                    pypi_0    pypi
astunparse                1.6.3                    pypi_0    pypi
ca-certificates           2021.10.26           haa95532_2
cached-property           1.5.2                    pypi_0    pypi
cachetools                4.2.4                    pypi_0    pypi
certifi                   2021.10.8        py37haa95532_0
charset-normalizer        2.0.9                    pypi_0    pypi
colorama                  0.4.4                    pypi_0    pypi
cycler                    0.11.0                   pypi_0    pypi
flatbuffers               2.0                      pypi_0    pypi
fonttools                 4.28.3                   pypi_0    pypi
gast                      0.4.0                    pypi_0    pypi
google-auth               2.3.3                    pypi_0    pypi
google-auth-oauthlib      0.4.6                    pypi_0    pypi
google-pasta              0.2.0                    pypi_0    pypi
grpcio                    1.42.0                   pypi_0    pypi
h5py                      3.6.0                    pypi_0    pypi
idna                      3.3                      pypi_0    pypi
importlib-metadata        4.8.2                    pypi_0    pypi
keras                     2.7.0                    pypi_0    pypi
keras-preprocessing       1.1.2                    pypi_0    pypi
kiwisolver                1.3.2                    pypi_0    pypi
libclang                  12.0.0                   pypi_0    pypi
```

*Figure 1: Conda Environment*

Now, we can install our required dependencies on this environment. We can use python package management system PIP to install and manage software packages. Below is the overview of the packages required to run this project

**Basic**: Numpy, Pillow, Pandas, Matplotlib

**Image Processing**: OpenCV

**AI Framework**: Pytorch

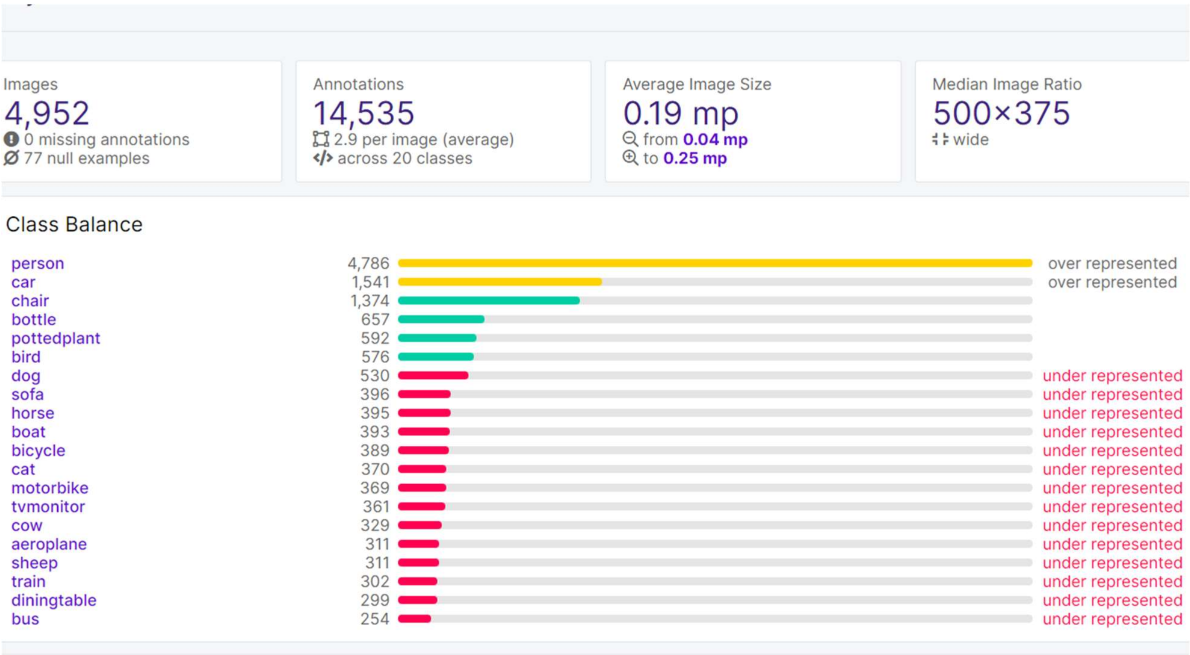
**GPU Toolkit**: CUDA

**Training Results**: Tensorboard

All these libraries with their version number are listed under *requirement.txt (Appendix 6.1)* file. We can use this file to install all dependencies at once.

*pip install -r requirements.txt*

## 3.2 Dataset Analysis



Images
4,952
0 missing annotations
77 null examples

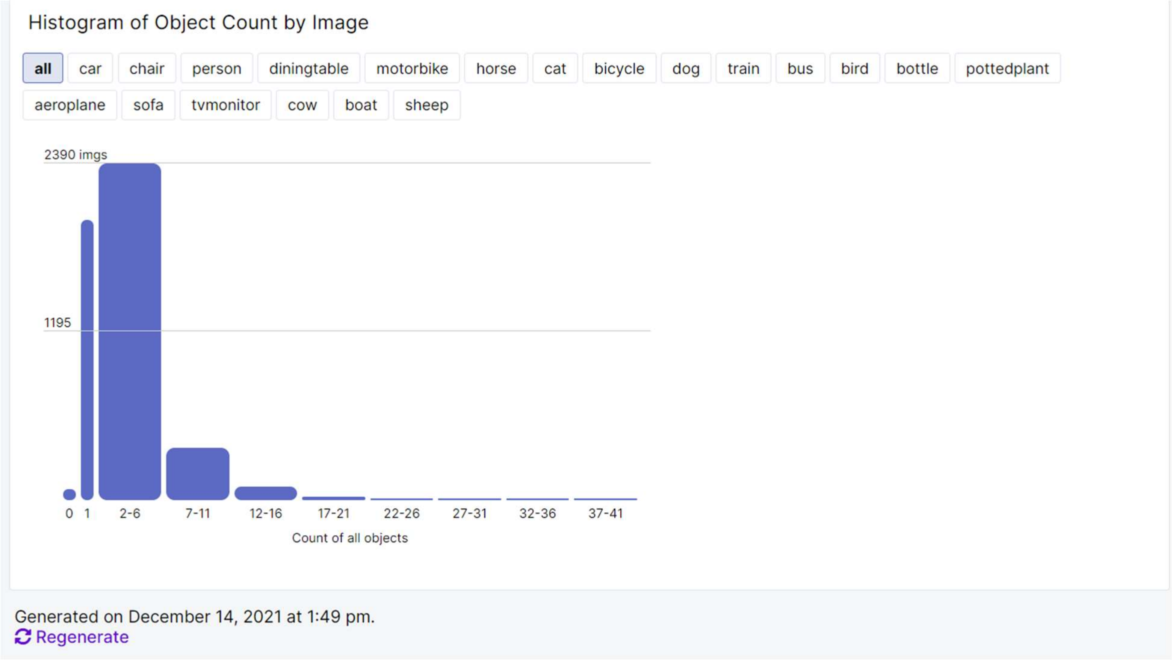
Annotations
14,535
2.9 per image (average)
across 20 classes

Average Image Size
0.19 mp
from **0.04 mp**
to **0.25 mp**

Median Image Ratio
500×375
wide

Class Balance

| | | |
|---|---|---|
| person | 4,786 | over represented |
| car | 1,541 | over represented |
| chair | 1,374 | |
| bottle | 657 | |
| pottedplant | 592 | |
| bird | 576 | |
| dog | 530 | under represented |
| sofa | 396 | under represented |
| horse | 395 | under represented |
| boat | 393 | under represented |
| bicycle | 389 | under represented |
| cat | 370 | under represented |
| motorbike | 369 | under represented |
| tvmonitor | 361 | under represented |
| cow | 329 | under represented |
| aeroplane | 311 | under represented |
| sheep | 311 | under represented |
| train | 302 | under represented |
| diningtable | 299 | under represented |
| bus | 254 | under represented |



Histogram of Object Count by Image

Generated on December 14, 2021 at 1:49 pm.

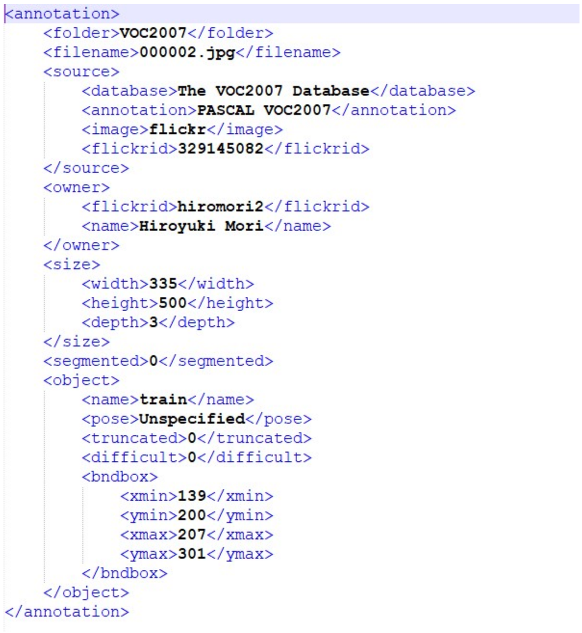
## 3.3 Dataset Preprocessing

Dataset Preprocessing is required to use that dataset before training. It includes below steps,

- Image Annotations
- Image Augmentations
- Resizing Images etc.

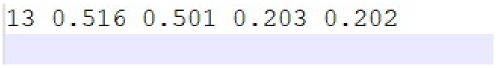
In this project we are using Pascal VOC 2007 dataset. Since this dataset is already annotated, we can skip first two steps. But this dataset comes with xml and Jason annotations and YOLO algorithm use its own annotation format. So first we need to convert annotation file of all images from xml to yolo format.

Below example shows image annotation in xml format.



```
<annotation>
    <folder>VOC2007</folder>
    <filename>000002.jpg</filename>
    <source>
        <database>The VOC2007 Database</database>
        <annotation>PASCAL VOC2007</annotation>
        <image>flickr</image>
        <flickrid>329145082</flickrid>
    </source>
    <owner>
        <flickrid>hiromori2</flickrid>
        <name>Hiroyuki Mori</name>
    </owner>
    <size>
        <width>335</width>
        <height>500</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>train</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>139</xmin>
            <ymin>200</ymin>
            <xmax>207</xmax>
            <ymax>301</ymax>
        </bndbox>
    </object>
</annotation>
```

But in this project below YOLO Annotation Format required.



```
13 0.516 0.501 0.203 0.202
```

Understanding YOLO format:

**13**: Dataset Object ID (in our case 20 objects are there)
**0.516**: Normalized X Coordinate of the Centre of Bounding Box
**0.501**: Normalized Y Coordinate of the Centre of Bounding Box
**0.203**: Normalized Width of the Bounding Box
**0.202**: Normalized Height of the Bounding Box

**Image Resizing** :

We need to resize all the images in dataset from 500x375 to 320x320 or 640x640 to make them compatible with Yolov5 models provided by Ultralytics. We can use Pillow library for resizing all the images.

**Syntax:** *Image.resize(size,resample=0)*
**size***: The requested size in pixels, as a 2-tuple: (width, height).*
**Resample:** *An optional resampling filter*

**Dataset Structure**:

Ultralytics has provided framework to train and test Yolov5 models. To use this framework, we need create dataset folder structure as required. We need to split data as 70% Training Data, 20% Testing Data and 10% Validation Data.
Dataset : **9963** Images
Training Data: **5928** Images
Testing Data: **3043** Images
Validation Data: **992** Images

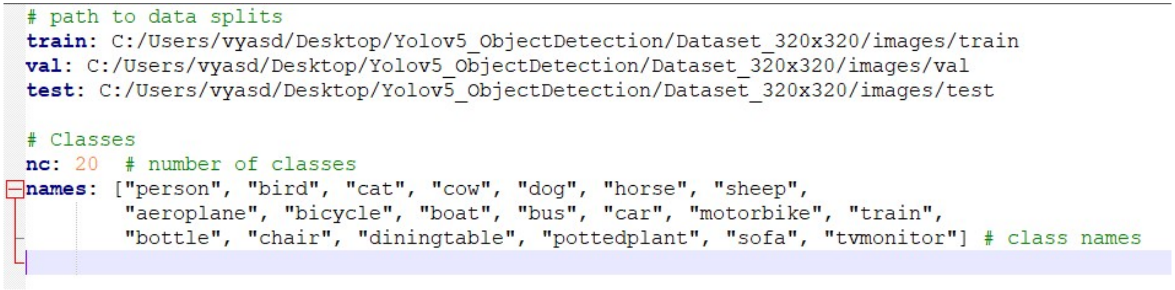
Dataset_320x320

| images | labels |
|---|---|
| test | test |
| 003905.jpg | 003905.txt |
| 003907.jpg | 003907.txt |
| . | . |
| . | . |
| 009961.jpg | 009961.txt |
| train | train |
| 000001.jpg | 000001.txt |
| 000002.jpg | 000002.txt |
| . | . |
| . | . |
| 007983.jpg | 007983.txt |
| val | val |
| 007985.jpg | 007985.txt |
| 007986.jpg | 007986.txt |
| . | . |
| . | . |
| 009963.jpg | 009963.txt |

## 3.4 Training Configuration

Yolov5 Model take input YAML file for training and testing of model. YAML file Train/Test/Val Dataset folder file paths. Also, it has label map which tells the training file what each object is by defining a mapping of class names to class ID numbers.

Below example of Pascal_voc_2007.yaml that we need to create before training.



```yaml
# path to data splits
train: C:/Users/vyasd/Desktop/Yolov5_ObjectDetection/Dataset_320x320/images/train
val: C:/Users/vyasd/Desktop/Yolov5_ObjectDetection/Dataset_320x320/images/val
test: C:/Users/vyasd/Desktop/Yolov5_ObjectDetection/Dataset_320x320/images/test

# Classes
nc: 20  # number of classes
names: ["person", "bird", "cat", "cow", "dog", "horse", "sheep",
        "aeroplane", "bicycle", "boat", "bus", "car", "motorbike", "train",
        "bottle", "chair", "diningtable", "pottedplant", "sofa", "tvmonitor"] # class names
```

***Note***: "names" array should have same class ID index as given in YOLO annotation format.

**Download Pretrained Weights:**

Before Training we need to download model file pretrained on coco dataset. To download these weights, run below script in yolov5 folder.

```
./script- download_weights.sh -here.sh
```

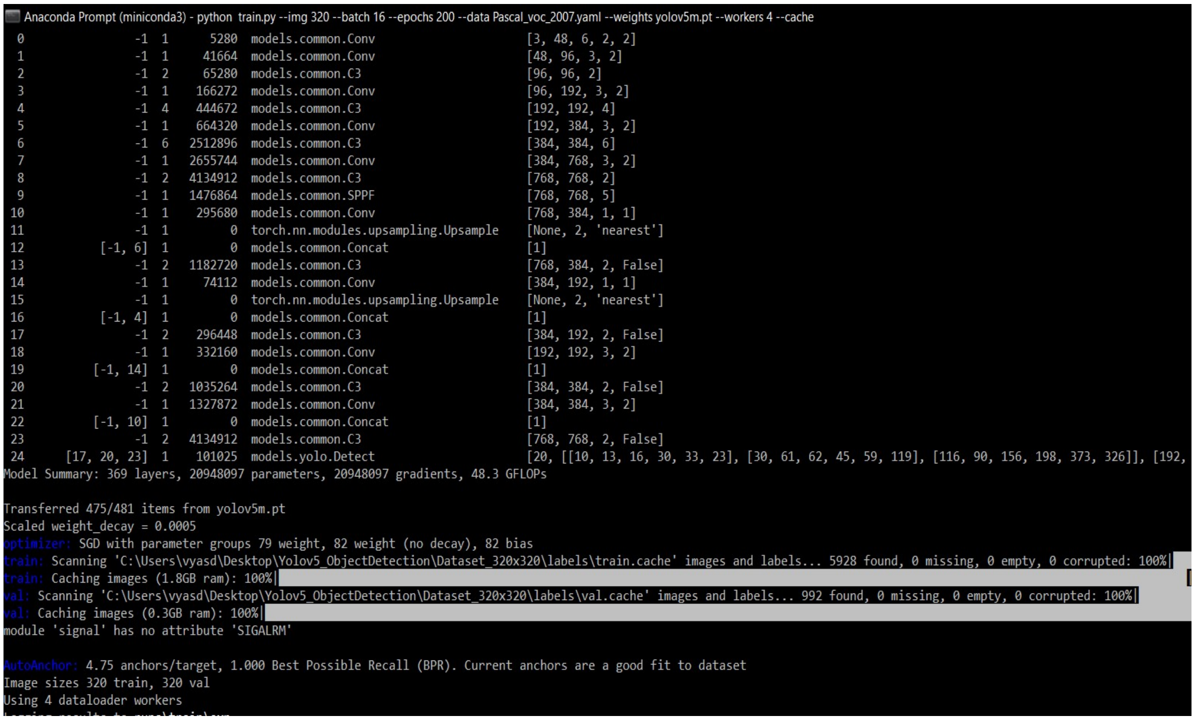
After executing this command all weights will be downloaded in your yolov5 project folder.
Weights: yolov5s.pt, yolov5m.pt, yolov5l.pt etc.

**Run Training**:

From yolov5 directory run following command in conda environment to begin model training:

```
python train.py --img 320 --batch 16 --epochs 200 --data Pascal_voc_2007.yaml --weights yolov5m.pt --workers 4 –cache
```

--img 320: (Image size 320x320)
--batch 16: (16 Images will be given to input at the same time to model)
--epochs 200: (Total 200 iterations will be run on entire train dataset)
--data Pascal_voc_2007.yaml: (Path to Training configuration file)
--weights yolov5m.pt (Path to YOLOv5 Model)



```
Anaconda Prompt (miniconda3) - python  train.py --img 320 --batch 16 --epochs 200 --data Pascal_voc_2007.yaml --weights yolov5m.pt --workers 4 --cache
  0                -1  1      5280  models.common.Conv                      [3, 48, 6, 2, 2]
  1                -1  1     41664  models.common.Conv                      [48, 96, 3, 2]
  2                -1  2     65280  models.common.C3                        [96, 96, 2]
  3                -1  1    166272  models.common.Conv                      [96, 192, 3, 2]
  4                -1  4    444672  models.common.C3                        [192, 192, 4]
  5                -1  1    664320  models.common.Conv                      [192, 384, 3, 2]
  6                -1  6   2512896  models.common.C3                        [384, 384, 6]
  7                -1  1   2655744  models.common.Conv                      [384, 768, 3, 2]
  8                -1  2   4134912  models.common.C3                        [768, 768, 2]
  9                -1  1   1476864  models.common.SPPF                      [768, 768, 5]
 10                -1  1    295680  models.common.Conv                      [768, 384, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12           [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  2   1182720  models.common.C3                        [768, 384, 2, False]
 14                -1  1     74112  models.common.Conv                      [384, 192, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16           [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  2    296448  models.common.C3                        [384, 192, 2, False]
 18                -1  1    332160  models.common.Conv                      [192, 192, 3, 2]
 19          [-1, 14]  1         0  models.common.Concat                    [1]
 20                -1  2   1035264  models.common.C3                        [384, 384, 2, False]
 21                -1  1   1327872  models.common.Conv                      [384, 384, 3, 2]
 22          [-1, 10]  1         0  models.common.Concat                    [1]
 23                -1  2   4134912  models.common.C3                        [768, 768, 2, False]
 24      [17, 20, 23]  1    101025  models.yolo.Detect                      [20, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [192,
Model Summary: 369 layers, 20948097 parameters, 20948097 gradients, 48.3 GFLOPs

Transferred 475/481 items from yolov5m.pt
Scaled weight_decay = 0.0005
optimizer: SGD with parameter groups 79 weight, 82 weight (no decay), 82 bias
train: Scanning 'C:\Users\vyasd\Desktop\Yolov5_ObjectDetection\Dataset_320x320\labels\train.cache' images and labels... 5928 found, 0 missing, 0 empty, 0 corrupted: 100%|
train: Caching images (1.8GB ram): 100%|
val: Scanning 'C:\Users\vyasd\Desktop\Yolov5_ObjectDetection\Dataset_320x320\labels\val.cache' images and labels... 992 found, 0 missing, 0 empty, 0 corrupted: 100%|
val: Caching images (0.3GB ram): 100%|
module 'signal' has no attribute 'SIGALRM'

AutoAnchor: 4.75 anchors/target, 1.000 Best Possible Recall (BPR). Current anchors are a good fit to dataset
Image sizes 320 train, 320 val
Using 4 dataloader workers
```

Pytorch will initialize the training, it usually takes few seconds for initialization before the actual training begins. Figure above shows command prompt after training begins.

We can also view the progress of the training job by using Tensorboard. To view training process, give following command on conda environment:

*tensorboard --logdir runs/train*

One of the important graphs is the Loss graph, which shows the overall loss of the classifier over time. The training process periodically saves checkpoints about every few minutes. This checkpoint will be saved as best.pt and last.pt. If training error occurred due to some unexpected reasons these checkpoints can be useful as we don’t have start training our model from the beginning which saves a lot of time.

## 3.5 Testing Model

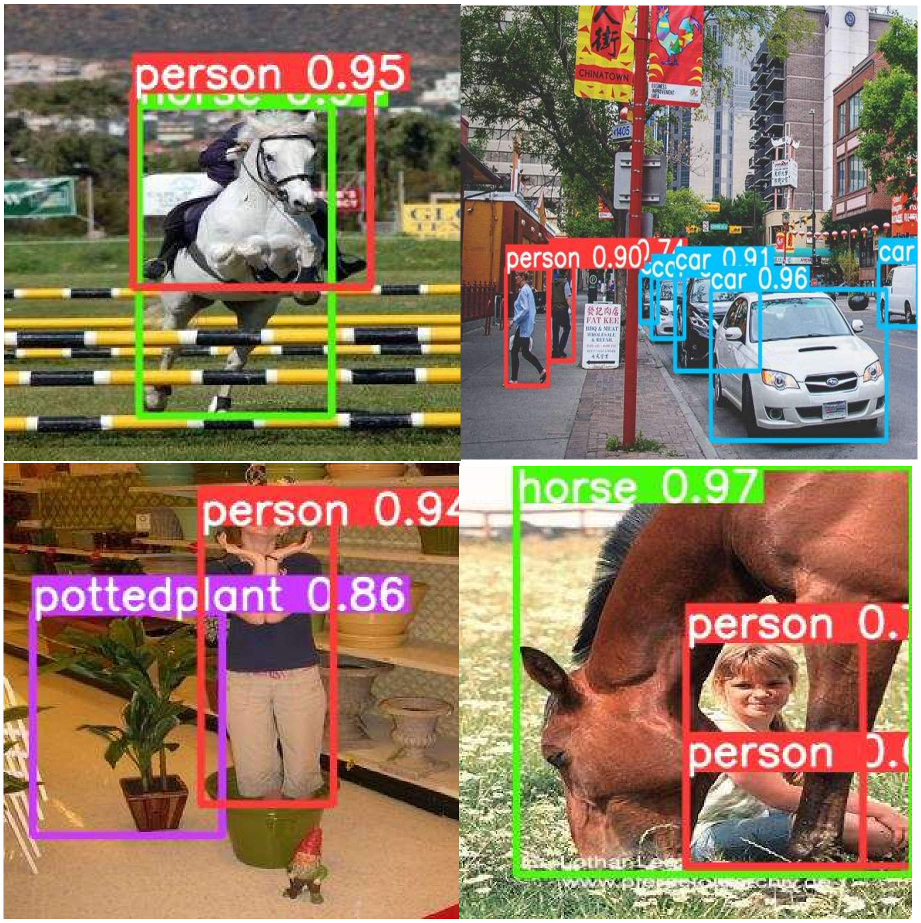
To test model on test dataset run following command on conda environment.

```
python test.py --img 320 --batch 16 --data Pascal_voc_2007.yaml --weights Yolov5m_PascalVoc2007.pt --workers 4
```

This command will generate confusion matrix and calculate overall mean average precision on test dataset. All results can be seen in /yolov5/runs/train directory.

We can use trained model to detect 20 objects in images, video, webcam etc. To detect objects run following command.

```
python detect.py --source /Yolov5_ObjectDetection/Dataset_320x320/images/test --weights /Yolov5_ObjectDetection/yolov5/Yolov5m_PascalVoc2007.pt --imgsz 320
```



To detect objects runtime from webcam use –source 0 in above command instead of file path.

# 4. Results

For comparing and testing different object detection model we can follow below parameters.

- Precision
  - mAP
- Recall
- Loss
  - Train Box Loss
  - Train Class Loss
  - Validation Box Loss
  - Validation Class Loss

**Precision**:

Precision is percentage measure that your predictions are correct.

$$Precision = \frac{TP}{TP + FP}$$

In our Case 20 classes are there. Therefore, Mean Average Precision will be more accurate measure.

$$mAP = \frac{1}{|classes|} \sum_{c \in classes} \frac{\#TP(c)}{\#TP(c) + \#FP(c)}$$

**Recall**:

Precision is percentage measures that how good a model is to find all the positives.

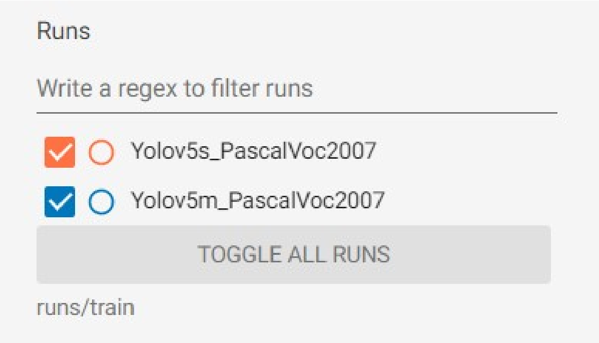
$$Recall = \frac{TP}{TP + FN}$$

$$mAR = \frac{1}{|classes|} \sum_{c \in classes} \frac{\#TP(c)}{\#TP(c) + \#FN(c)}$$

**TP**: True positive, **FN**: False Negative, **FP**: False Negative

### Yolov5m vs Yolov5s model comparison:

In below graphs blue line is represents **Yolov5m** model data and orange line represents **Yolov5s** model data.



Runs

Write a regex to filter runs

☑ ○ Yolov5s_PascalVoc2007

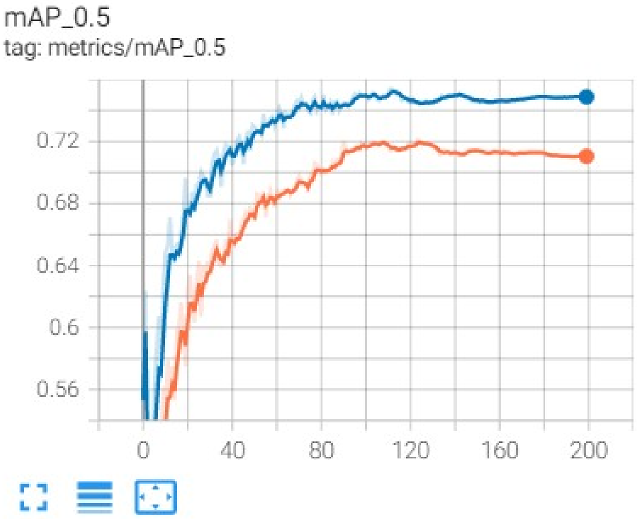
☑ ○ Yolov5m_PascalVoc2007

TOGGLE ALL RUNS

runs/train

### Training and Validation Results:

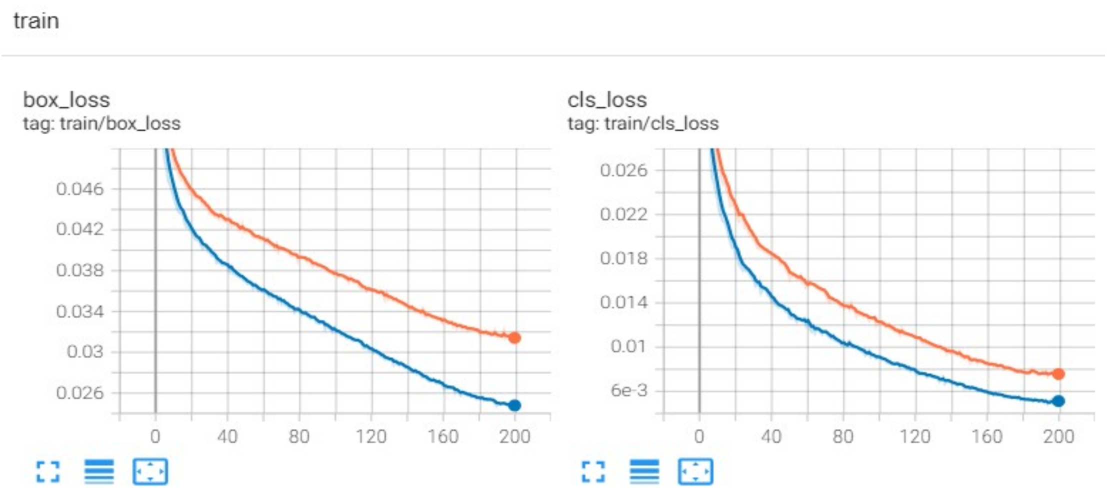
Mean Average Precision**:**



In above figure Y-Axis shows value of mAP and X-Axis shows number of epochs. As we can see from the graph model performed well on validation dataset as training progresses mAP increases.

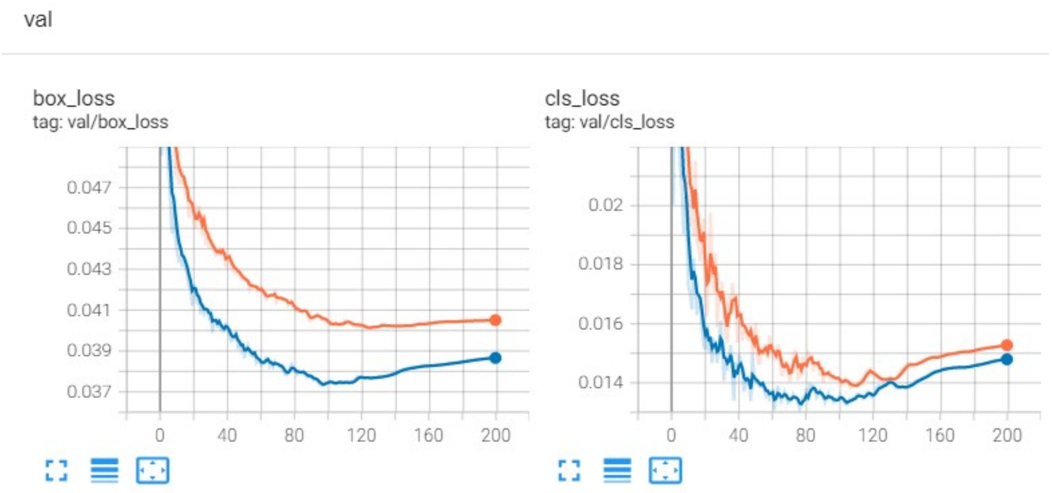
Model Yolov5m shows better mAP than Yolov5s since it has a greater number of layers and training parameters.

Training Losses:



Above figures show Training Losses during the training of both models. In above figure Y-Axis shows value of  box loss/class loss and X-Axis shows number of epochs. As we can see from the graph model performed well on training dataset as training progresses both the loss decreases.

Validation Losses:



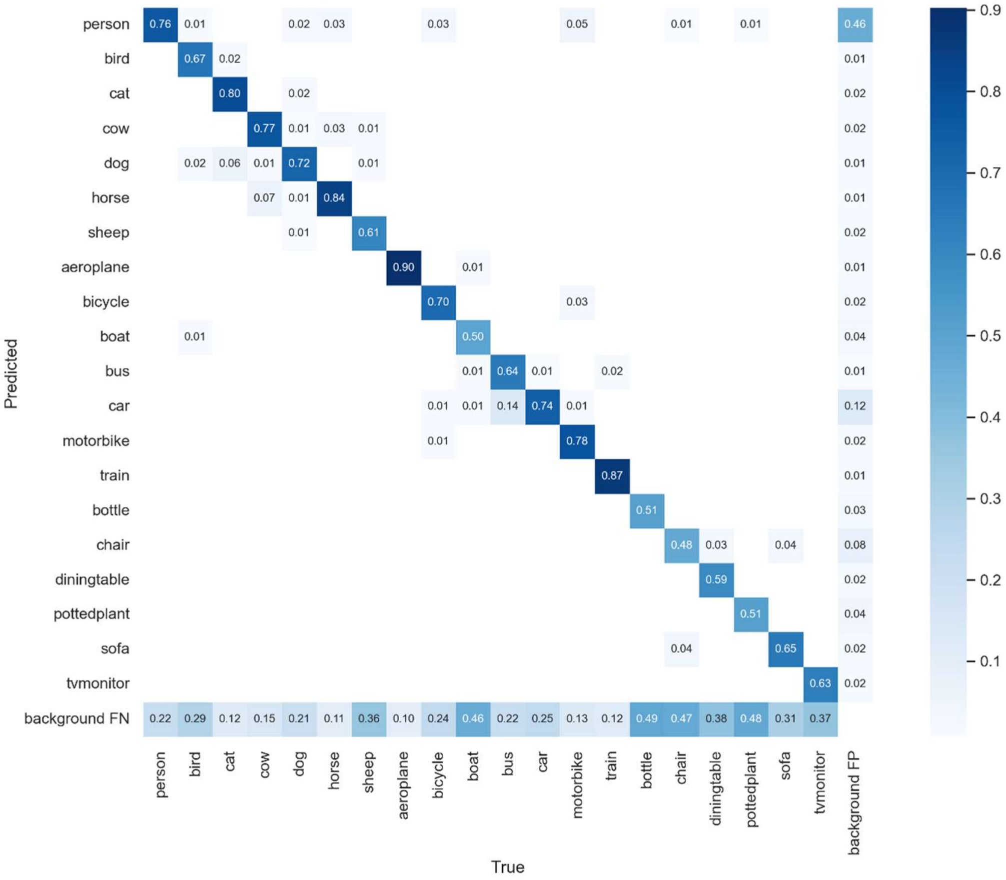
Above figures show Validation Losses during the training of both models. In above figure Y-Axis shows value of  box loss/class loss and X-Axis shows number of epochs. As we can see from the graph model performed well on validation dataset as training progresses both the loss decreases.

As we can see Validation losses are starting to increase after 120 epochs, we should not train model for more epochs to prevent model from overfitting.

## Testing Results:

Confusion Matrix is a table that is used to describe performance of the classification on Test dataset for which the true values are known.

## Yolov5m Confusion Matrix:

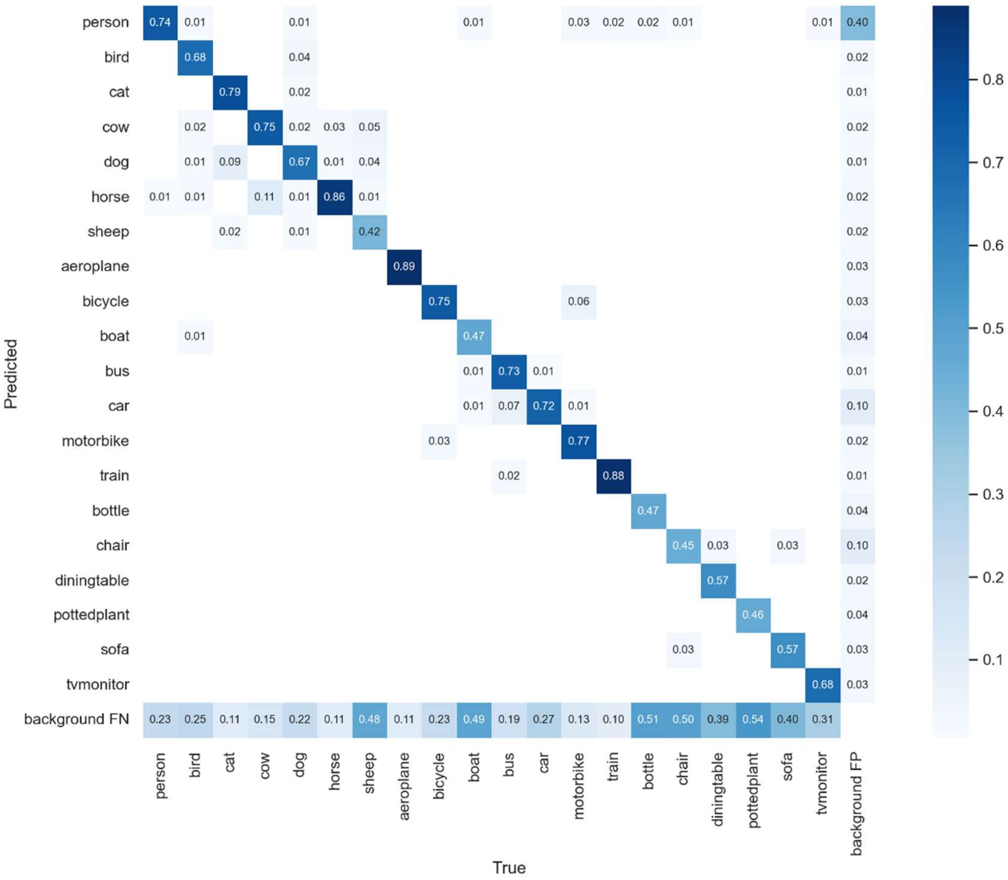


Above figure shows confusion matrix for Yolov5m model. In figure Y-Axis shows Predicted Labels and X-Axis shows True Labels for each 20 classes. By above confusion matrix is normalized with total number of labels for each class. At 1,1 position 0.67 means 67 labels of class "bird" are detected correctly out of 100. Other 0.29, 0.01, 0.02, 0.01 means below respectively,

- 29 labels of class "bird" out of 100 are not detected at all.
- 1 label of class "bird" out of 100 are detected as person.
- 2 labels of class "bird" out of 100 are detected as dog.
- 1 label of class "bird" out of 100 are detected as boat.

## Mean Average Precision(mAP) = 70.1%

## Yolov5s Confusion Matrix:



Above figure shows confusion matrix for Yolov5s model. In figure Y-Axis shows Predicted Labels and X-Axis shows True Labels for each 20 classes. By above confusion matrix is normalized with total number of labels for each class. At 1,1 position 0.68 means 68 labels of class “bird” are detected correctly out of 100.

### Mean Average Precision(mAP) = 68.3%

# 5. Conclusion

Trend in computer vison algorithms studied. Analysis of Pascal VOC 2007 dataset was carried out. Yolov5s and Yolov5m algorithms were selected for object detection task. Both models were trained on VOC 2007 dataset. Application was tested on test dataset and all model performance like mean average precision, Training Losses, Validation Losses, Confusion Matrix etc. were studied Yolov5m model performed better compared to Yolov5s on test dataset with mAP of 70.1% and 68.3% respectively.

# 6. Future Work

Model precision and accuracy can be further improved by more data and handling dataset more effectively. As in the data analysis we could see that dataset was poorly created with much more "person" labels than some other labels like bus, sheep etc. This effects our overall results as we can see on the confusion matrix as well. Classes which have less data has much less precision compared to other class which has more images. More Image annotation methods can be applied on dataset like rotation, scaling, translation etc to improve performance. Adding more layers and training parameters can also help improving model performance but it makes model heavier.

# 7. References

[1]https://github.com/ultralytics/yolov5

[2]https://www.analyticsvidhya.com/blog/2021/08/train-your-own-yolov5-object-detection-model/

[3]https://paperswithcode.com/dataset/pascal-voc-2007

[4]https://towardsdatascience.com/how-to-train-a-custom-object-detection-model-with-yolo-v5-917e9ce13208

[5]https://www.youtube.com/watch?v=vXYMJFBLzfc&list=WL&index=41&ab_c[hannel=O-Vision

[6]https://www.youtube.com/watch?v=2nR2e4J4ZaI&list=WL&index=40&ab_channel=KarndeepSingh