Object recognition using YoloV4 and YoloV4-tiny on a custom dataset

1. Understanding the dataset

For the given dataset, the observation is

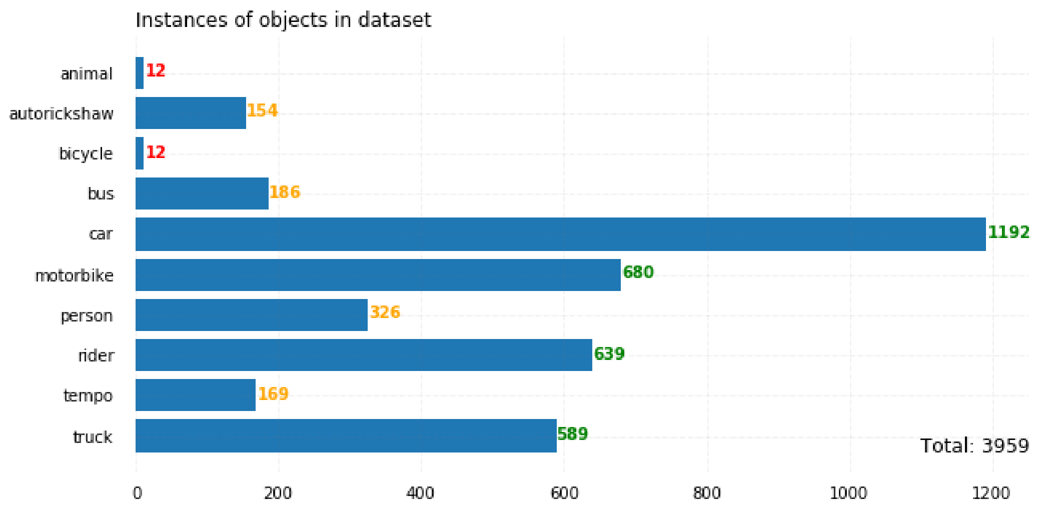
Number of classes: 10
- names: [animal, autorickshaw, bicycle, bus, car, motorbike, person, rider, tempo, truck]

Training data:
- Image resolution: 640 x 480
- Total number of training images: 1530

Test data:
- Image resolution: 1280 x 720
- Total number of test images: 250

1.1 Data preprocessing and cleaning

For the 1530 training images, 216 have no object or annotation. They can be used to provide the model with some negative samples to train it what not to learn from the data. This includes empty roads, trees, buildings or lights, etc.



Observations:

-Classes animal and bicycle are poorly represented with only 12 object instances each
-Autorickshaw, bus, person, and tempo have average representation
-Car is the most occurring object and is overrepresented.

-Number of training samples with object: 1314
-Total objects: 3959

The dataset was split in a **ratio of 80:20** for **training and validation** set creation. The images from the main set were split in random as maintaining class distribution is difficult as multiple objects are present in a single image. This process gave 1052 images for training and 262 images for validation.

1.2 Plausible results

1. The mAP of class animal and bicycle might come out to be the least of all due to less samples.
2. The model may overfit for the class car sample images
3. The accuracy might be affected for objects in low light as image resolution is less and features are pixelated.

2. Model selection

The ideal model should provide high average precision and be able to identify objects with very little latency. The current SOTA object detection models are YOLO, Faster-RCNN, SSD-Mobilenet or SSD-Resnet. The following limitations need to be considered.

1. Limitations of Faster-RCNN:

    -Architecture involves 2 stage feature learning and detection. This adds latency and inference speed is reduced.

    -Though precision is improved, it is not suitable for real-time object detection as it takes 47 sec for a single image.

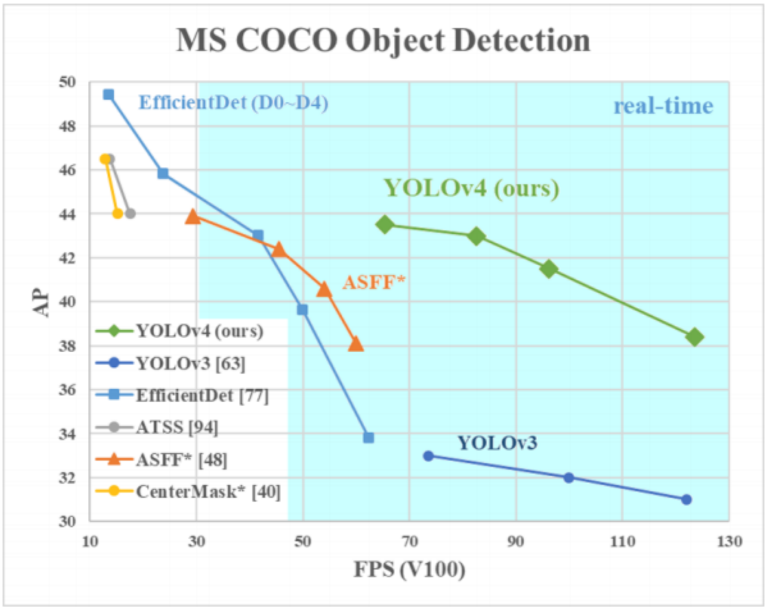
    - The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

2. Limitations of SSD:

    -Though architecture is a single shot learning and detection, it requires more negative samples for the network to identify what features are not of the object.

    -Since 2 classes (animal and bicycle) have very little data, adding negative samples will create uneven biases and these class features will not be learned accurately.

    -Moreover, SSD does not identify objects which have small areas well



AP vs FPS plot (from main YoloV4 paper)

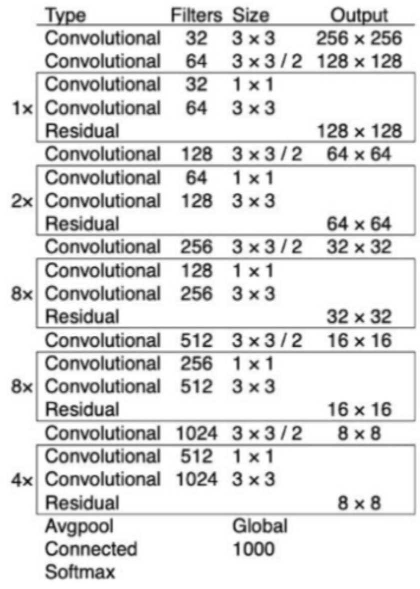
Hence, YOLO architecture provides the required inference speed as well as the precision for real-time object detection. It is robust to object shape and does not need large negative samples in the dataset. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. Of the varied versions of YOLO, YoloV4 is selected for the following reasons:

1. Dataset has small, medium, and large objects based on area and yolov4 performs well even on such a varied dataset.
2. The inference speed of YoloV4 is a good factor to use for real-time object detection.
3. Yolov4 still uses C at its core and hence is a little faster than the recent Yolov5 algorithm, which runs on python.
4. YoloV4 is based on CSPDarknet53 which is a modified Resnet53 CNN architecture. This enables good feature learning architecture for the YoloV4 algorithm.
5. YoloV4 preemptively performs data augmentation like (crop, rotation, flip, aspect, mosaic, blur, etc) hence preprocessing of data is not required much.
6. Yolov4 uses a genetic algorithm to select certain hyperparameters and hence can optimise within the solution space well.

YOLO v4 Architecture:

The YoloV4 architecture comprises of 3 parts:

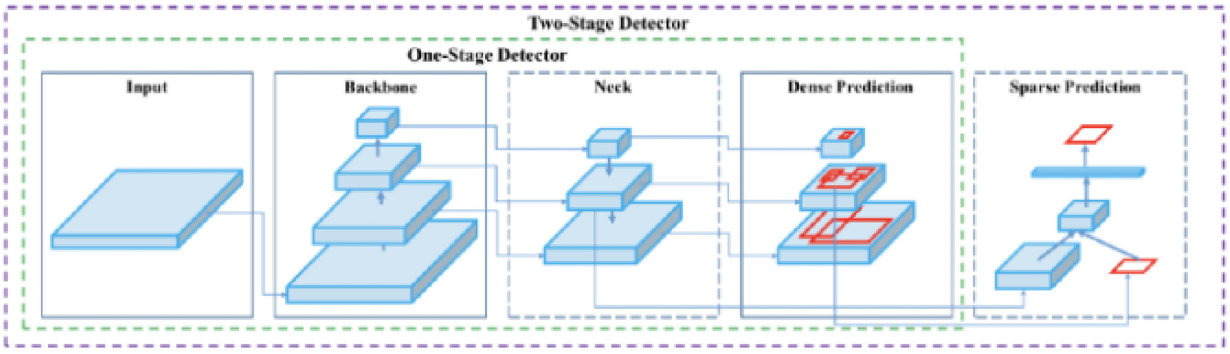
1. <u>Backbone</u>: This is the main feature detection architecture which uses CSP Darknet53 and has 53 convolutional layers. CSP is cross staged partial connections which separate the current layer into 2 parts, one that will go through a block of convolutions, and one that won't. Then, results are aggregated.



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

YoloV4 backbone

2. Neck: The purpose of the neck block is to add extra layers between the backbone and the head (dense prediction block). This adds more features to learn and increases feature space.
3. Head (dense prediction layer): The main purpose is to identify the bounding boxes and classify what is inside the box.



Architecture of object detection algorithms

The architecture highlighted in green box is used in YoloV4. It is a single stage detection algorithm which makes it the optimal object detection algorithm for real time applications.

In addition to the architecture, YoloV4 introduced BoF(Bag of freebies) and BoS(Bag of specials). BoF will gather techniques such as dropout and data augmentation, while BoS involves a neck, parameter aggregation, and specific techniques for activation, non-maxima suppression, and more.

3. Approach

To identify which hyperparameter value suits the dataset well and gives the maximum precision, we follow a bottom up approach where we keep all the values default and try changing them after every training run. In addition, a smaller version of YoloV4 *called YoloV4 tiny* is also trained which can be used in constrained and edge devices. The Yolov4 tiny is less accurate as it uses only 2 yolo blocks, but the latency and model size is reduced greatly. In the end we will have the flexibility to use any model based on what is our priority, precision or inference time. **For the current task, the main YoloV4 is used as it's precision is higher and latency (when both models were run on a GPU) is on similar lines**. YoloV4 tiny is trained just to experiment the precision vs latency tradeoff for the object detection models. This is discussed further in the results section.

The darknet framework is used to train the models and it saves weight every 1000 epochs. So we have flexibility to use the weight file for which loss is least and mAP is high.

Since the model is now finalized, we have to put our custom data to use and train the YoloV4 model on this dataset.

The steps involved in this process are:

1. Convert the COCO Json annotations to the YOLO txt annotations for each image. This is done using the json2txt.py code.
2. Split the dataset in 80:20 train:validation set so that a training loss and mAP can be calculated. Text files containing training and validation image paths are also created.
3. Configure the required files for the Yolo object detection algorithm. Obj.names contains a list of all the class labels. Obj.data contains information about training and validation images and backup weights folder.
4. Train the model on custom data using the pretrained weights. Once finished we can evaluate the model and change the parameters to improve precision.
5. Use *cv2.dnn.readNetFromDarknet()* function to get inferences.

Steps identified to increase precision:

1. Setting *random=1*. This increased precision by training Yolo for different image resolutions.
2. Increasing input image size. This enabled more features to be identified.
3. Setting *ignore_thresh=0.9* and *iou_loss=giou* will increase mAP at 0.9 but it decreased mAP at 0.5. (These changes were not implemented for the final model as mAP at 0.5 provides optimal bounding boxes and class confidences.)
4. Add negative samples to the training set to provide with features the model does not have to learn

4. Hyperparameters and test settings

4.1 Training hyperparameters (YOLOv4):

Batch size = 64	Activation = mish, leaky relu
Subdivisions = 16	Conv Filter, stride, pad = (3,1,1)
Input image size = 416 x 416	Epochs = 20000
Learning rate = 0.001	IoU loss = CIoU
Decay = 0.0005	Nms suppression = greedynms
Momentum = 0.949	Random = 1
Saturation, exposure, hue = (1.5,1.5,0.1)	

4.2 Training hyperparameters (YOLOv4-tiny):

Batch size = 64
Subdivisions = 8
Input image size = 608 x 608
Learning rate = 0.001
Decay = 0.0005
Momentum = 0.949
Saturation, exposure, hue = (1.5,1.5,0.1)

Activation = leaky relu
Conv Filter, stride, pad = (3,1,1)
Epochs = 20000
IoU loss = CIoU
Random = 1

4.3 Testing/Inference settings:

For the inference purpose, the following parameters are changed to get the best bounding boxes.

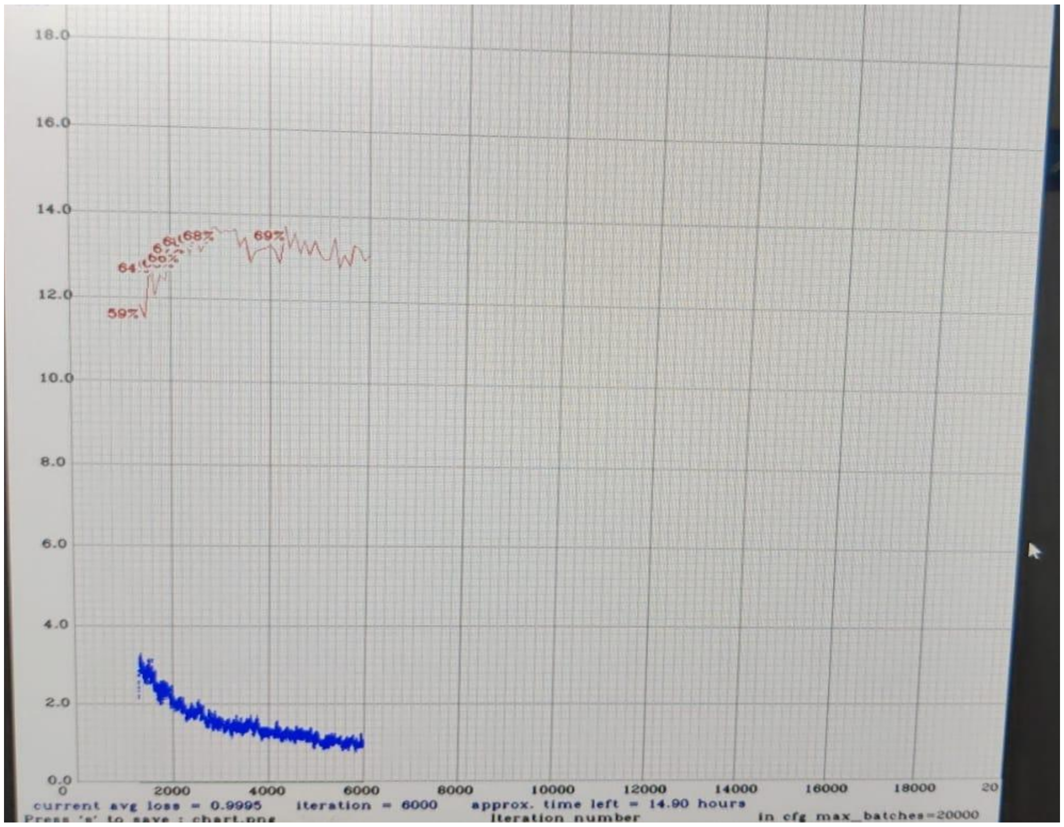
Input image size: 416 x416
NMS threshold: 0.3 (removes all bounding boxes with less than 30% confidence)
Conf Threshold: 0.5 (From final bounding box, show the ones with confidence score > 50%)
Decimal precision: 2

5. Training log and loss plots

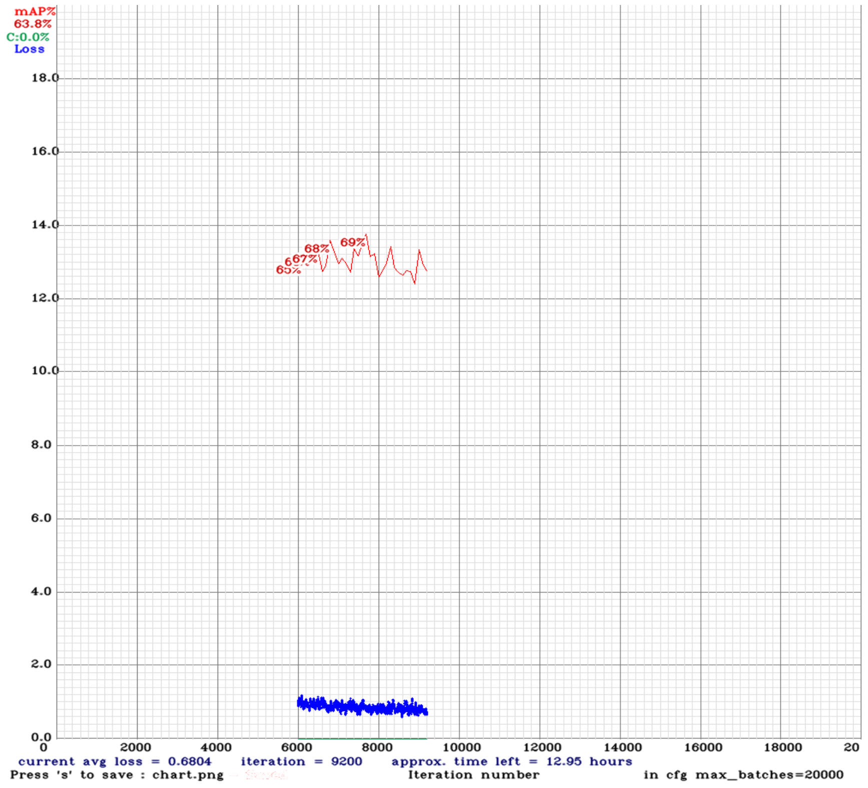
NOTE: Since the YoloV4 has more layers and requires the min number of epochs as (num_classes*2000), the training ran for a long time and had to be done in breaks, due to colab timeouts. Hence, there are 5 sections of the graph for loss and mAP, and log file is not available for this as it was over written on every resume point. However, the loss and mAP for every 1000 epoch can be found from **Step 9** of the colab file *(DBAI Model Training steps)*.

5.1 YoloV4 plots:

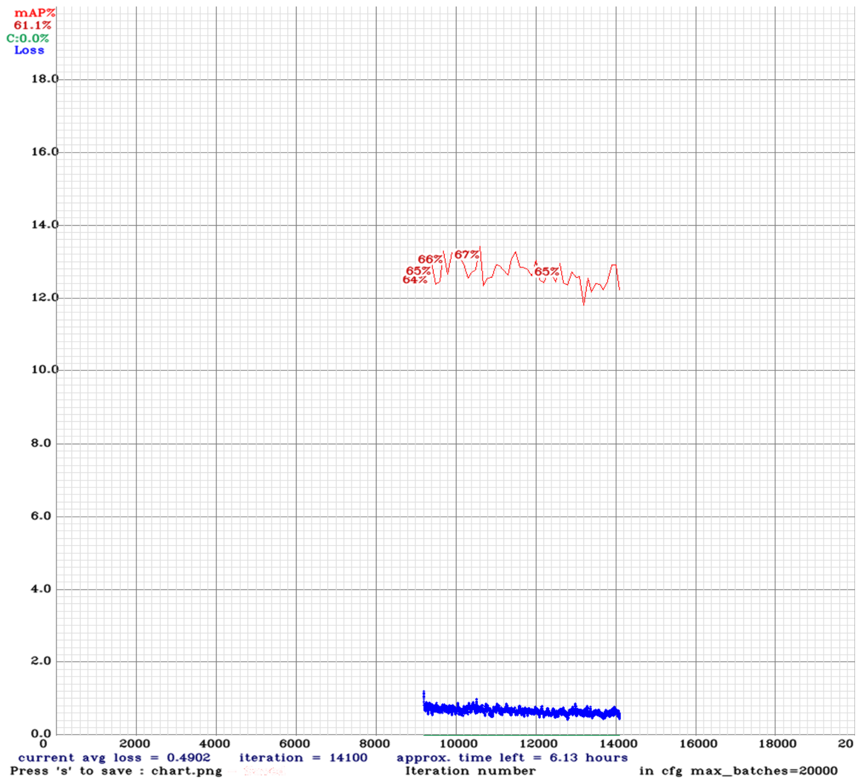
The mAP scores are calculated at every 100 steps and hence the plots are not smooth. Whereas the loss is calculated after every step and the plot is very smooth.



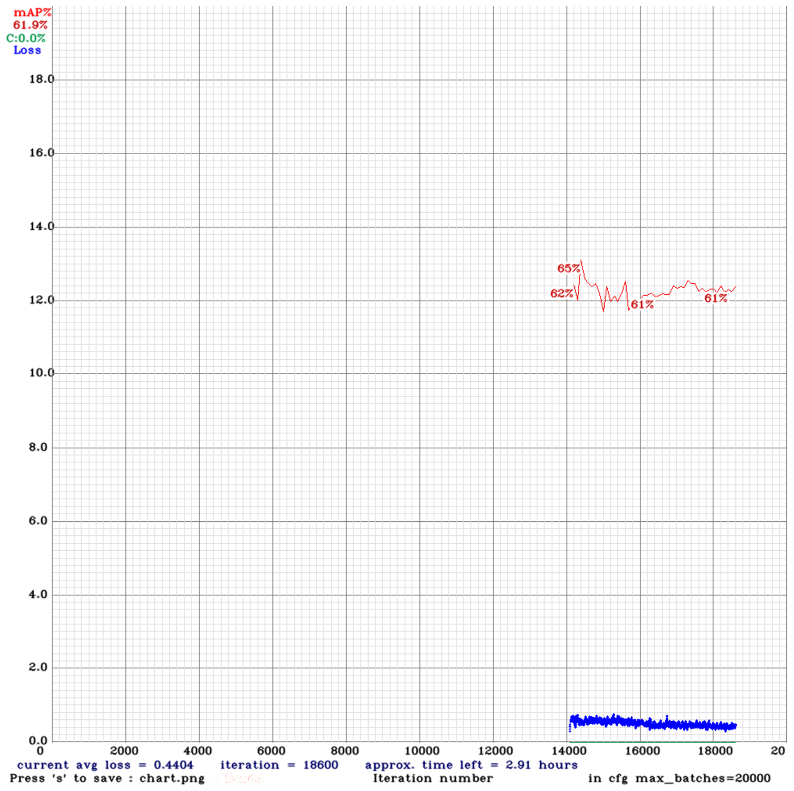
Pt 1 - Epoch 1000 to 6000



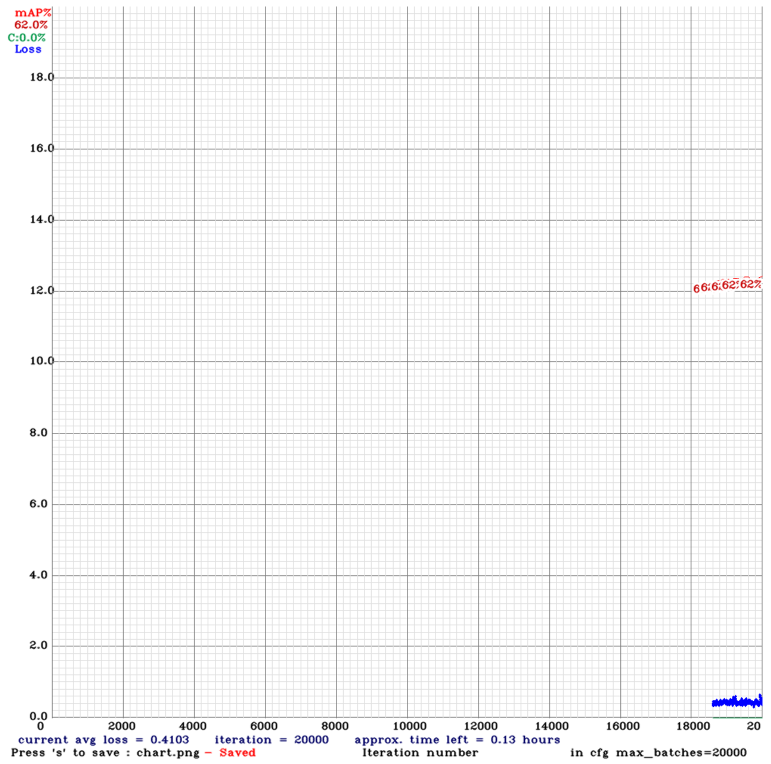
Pt 2 - Epoch 6000 to 9200



Pt 3 - Epoch 9200 to 14100



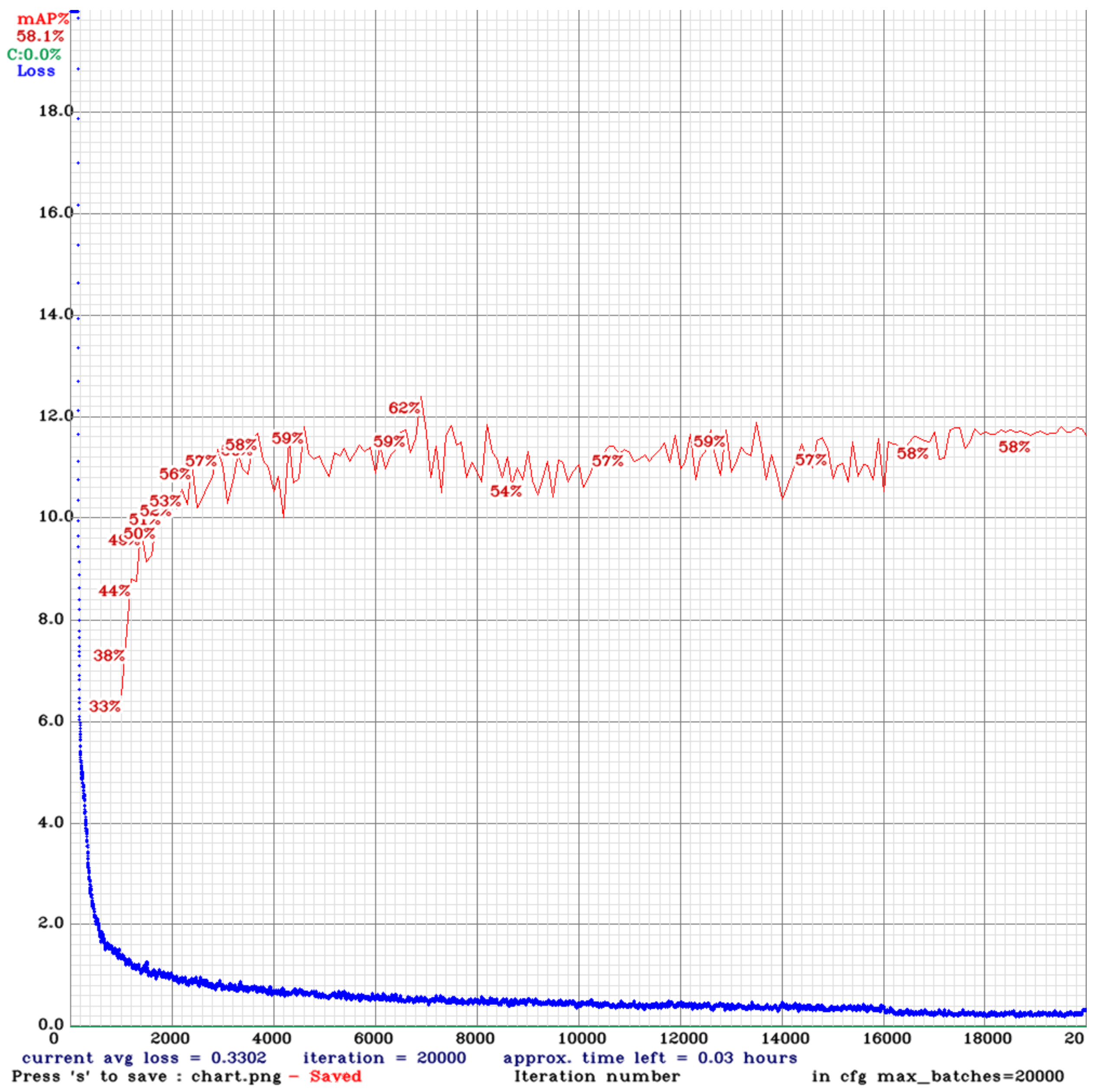
Pt 4 - Epoch 14100 to 18600



```
current avg loss = 0.4103    iteration = 20000    approx. time left = 0.13 hours
Press 's' to save : chart.png - Saved                                    in cfg max_batches=20000
```

Pt 5 - Epoch 18600 to 20000

<u>5.2 YoloV4-Tiny plot:</u>

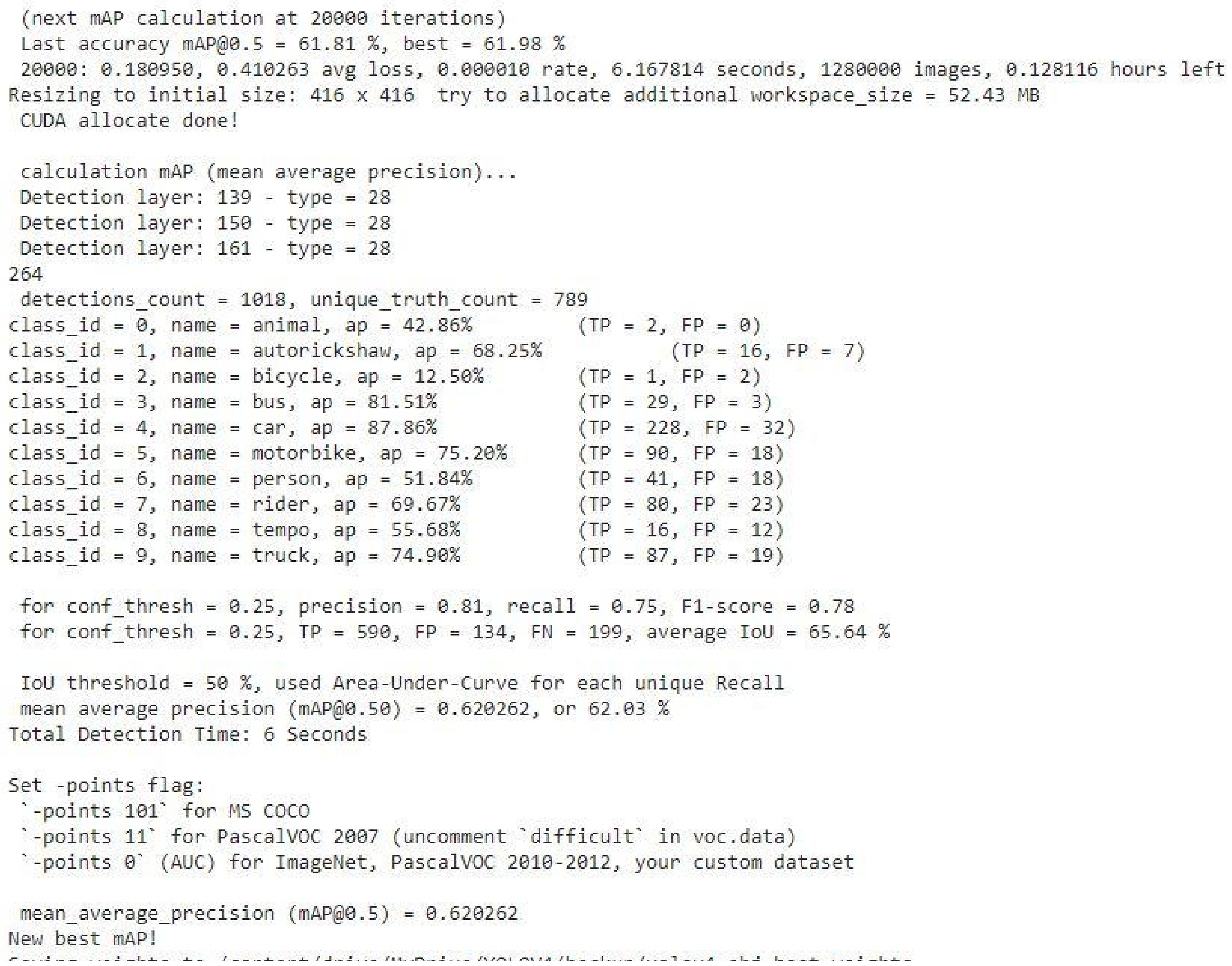


The training log for yoloV4-Tiny is present in the deliverables folder.

6. Results

6.1 YoloV4:

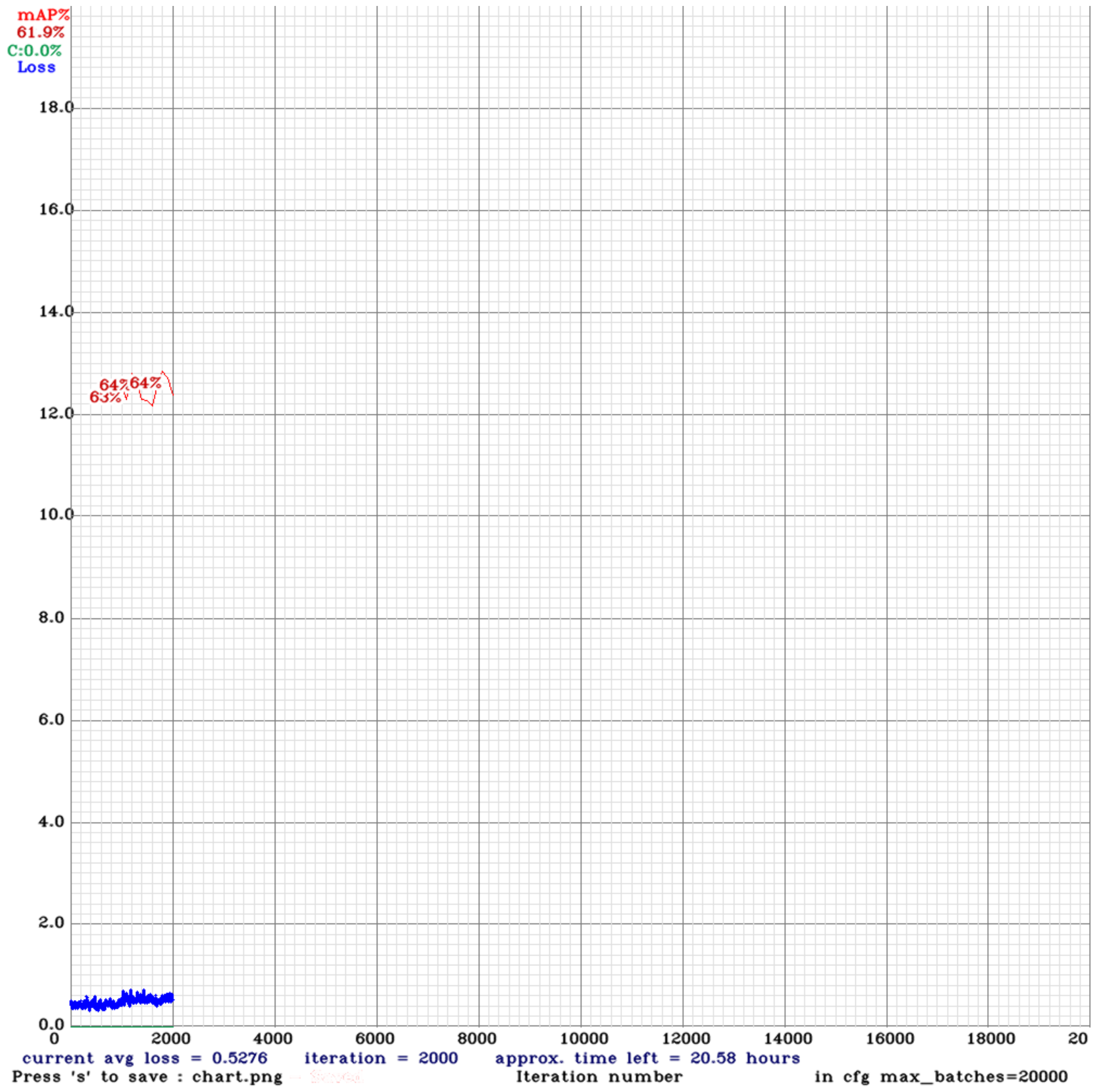
After 20000 epochs,



```
 (next mAP calculation at 20000 iterations)
 Last accuracy mAP@0.5 = 61.81 %, best = 61.98 %
 20000: 0.180950, 0.410263 avg loss, 0.000010 rate, 6.167814 seconds, 1280000 images, 0.128116 hours left
Resizing to initial size: 416 x 416  try to allocate additional workspace_size = 52.43 MB
 CUDA allocate done!

 calculation mAP (mean average precision)...
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
264
 detections_count = 1018, unique_truth_count = 789
class_id = 0, name = animal, ap = 42.86%         (TP = 2, FP = 0)
class_id = 1, name = autorickshaw, ap = 68.25%           (TP = 16, FP = 7)
class_id = 2, name = bicycle, ap = 12.50%        (TP = 1, FP = 2)
class_id = 3, name = bus, ap = 81.51%            (TP = 29, FP = 3)
class_id = 4, name = car, ap = 87.86%            (TP = 228, FP = 32)
class_id = 5, name = motorbike, ap = 75.20%      (TP = 90, FP = 18)
class_id = 6, name = person, ap = 51.84%         (TP = 41, FP = 18)
class_id = 7, name = rider, ap = 69.67%          (TP = 80, FP = 23)
class_id = 8, name = tempo, ap = 55.68%          (TP = 16, FP = 12)
class_id = 9, name = truck, ap = 74.90%          (TP = 87, FP = 19)

 for conf_thresh = 0.25, precision = 0.81, recall = 0.75, F1-score = 0.78
 for conf_thresh = 0.25, TP = 590, FP = 134, FN = 199, average IoU = 65.64 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.620262, or 62.03 %
Total Detection Time: 6 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

 mean_average_precision (mAP@0.5) = 0.620262
New best mAP!
```

- After 20000 epochs, the **mAP achieved is 62.03% and the loss is 0.41.**
- The classes animal and bicycle have a fairly low precision value compared to the other classes.
- The class car has the highest precision as it was the most represented class.

After continuing training for 2000 more epochs the following results were obtained:

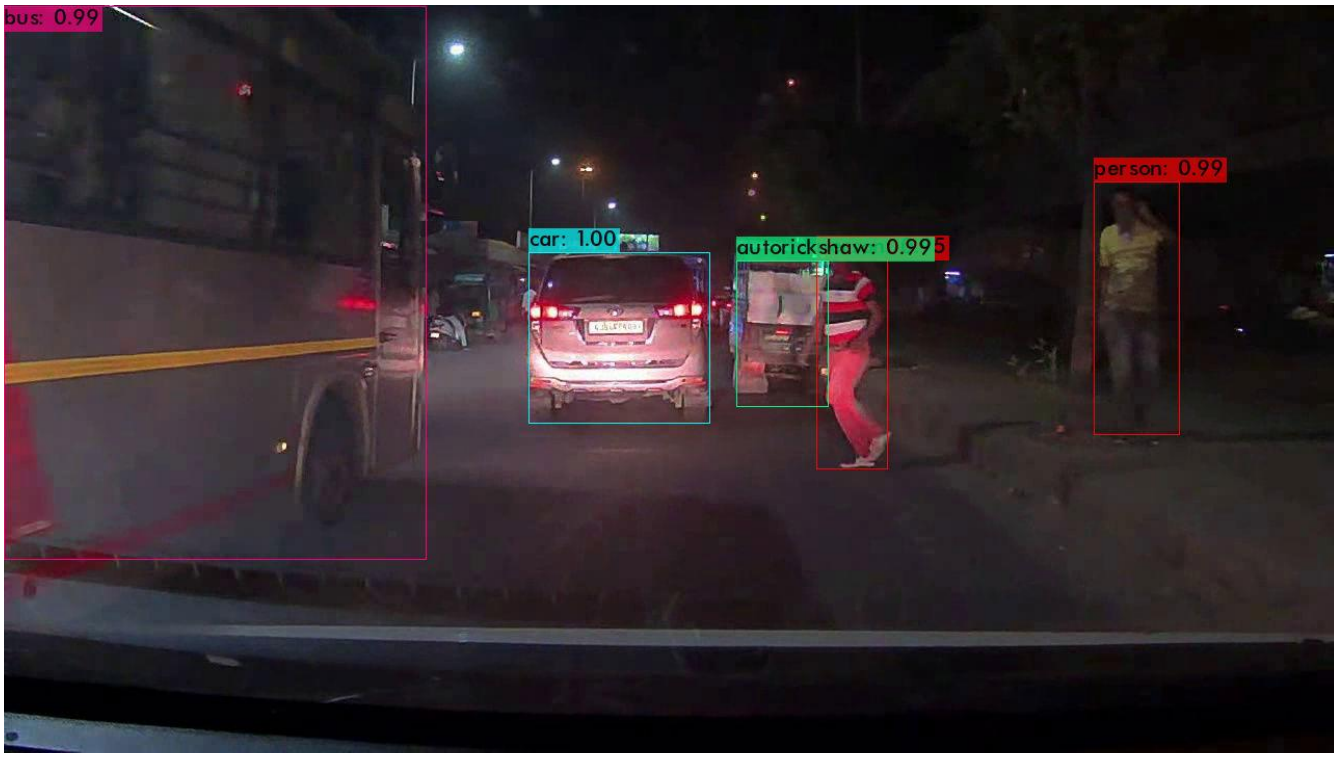


This final epoch run has the best mAP value for the least loss among all others. The training was continued after 20000 epochs to check if the model can converge any further. The following loss and mAP plot showed that after 20000 epochs the loss increased to 0.527. This indicated that the model is overfit and should not be continued further.

Hence, the final weights after 20000 epochs will be used for prediction tasks.

The inference speed of YoloV4 is **20ms** on a GPU:

```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 59.628
avg_outputs = 490961
 Allocate additional workspace_size = 52.43 MB
Loading weights from /content/drive/MyDrive/YOLOV4/backup/yolov4-obj_final.weights...
 seen 64, trained: 1280 K-images (20 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
/content/drive/MyDrive/YOLOV4/video_rms_2020-11-07_18-54_128.jpg: Predicted in 20.583000 milli-seconds.
bus: 98.866920%
car: 99.972466%
autorickshaw: 99.104988%
person: 94.987877%
person: 98.561684%
```

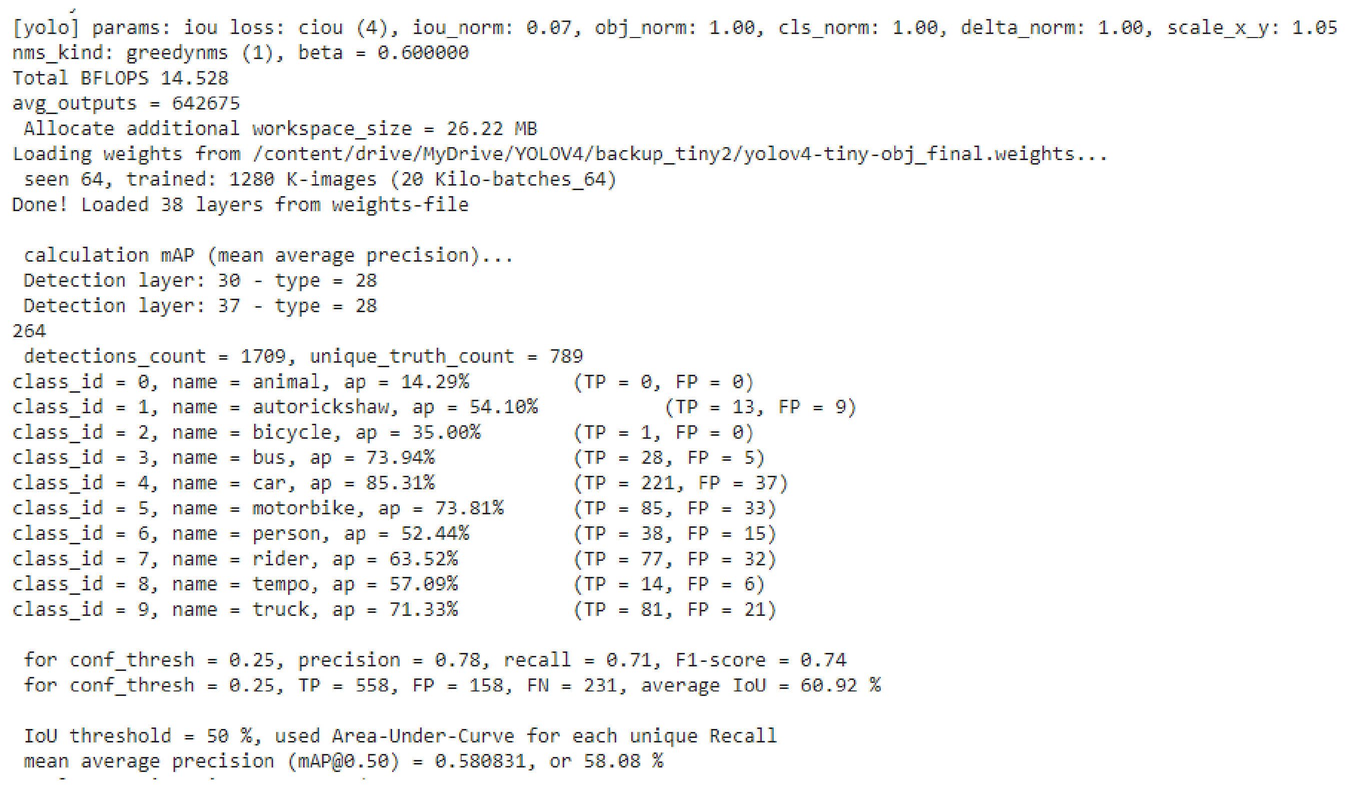


The model is able to detect 5 objects in the sample image. For this model precision is taken as the deciding factor. The aim is to detect objects with high accuracy.

6.2 YoloV4 - Tiny:

The main purpose of a second model is to experiment with the precision and inference time. The YoloV4 tiny provides a very less response time in the same setting as the YoloV4.

After 20000 epochs,

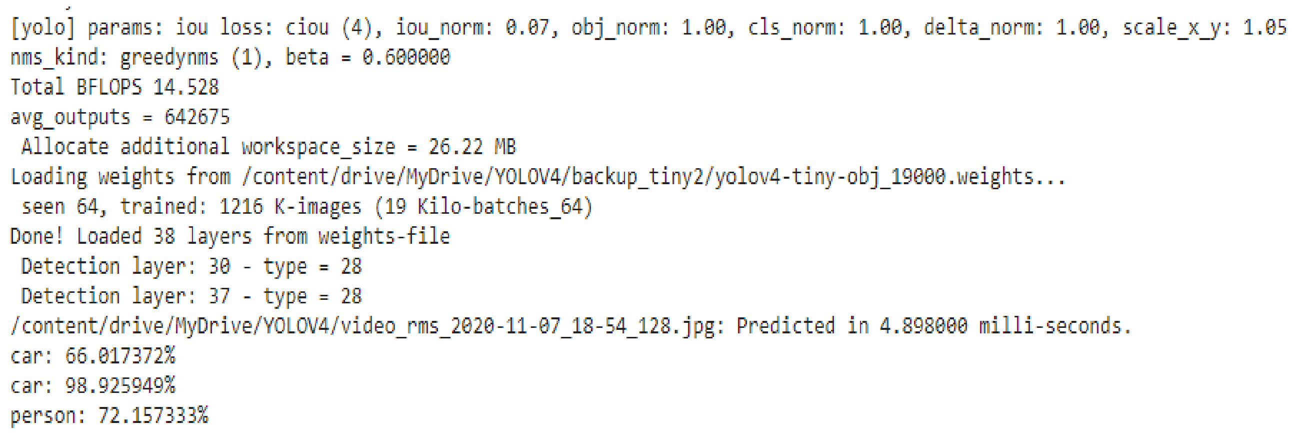


```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 14.528
avg_outputs = 642675
 Allocate additional workspace_size = 26.22 MB
Loading weights from /content/drive/MyDrive/YOLOV4/backup_tiny2/yolov4-tiny-obj_final.weights...
 seen 64, trained: 1280 K-images (20 Kilo-batches_64)
Done! Loaded 38 layers from weights-file

 calculation mAP (mean average precision)...
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
264
 detections_count = 1709, unique_truth_count = 789
class_id = 0, name = animal, ap = 14.29%        (TP = 0, FP = 0)
class_id = 1, name = autorickshaw, ap = 54.10%          (TP = 13, FP = 9)
class_id = 2, name = bicycle, ap = 35.00%       (TP = 1, FP = 0)
class_id = 3, name = bus, ap = 73.94%           (TP = 28, FP = 5)
class_id = 4, name = car, ap = 85.31%           (TP = 221, FP = 37)
class_id = 5, name = motorbike, ap = 73.81%     (TP = 85, FP = 33)
class_id = 6, name = person, ap = 52.44%        (TP = 38, FP = 15)
class_id = 7, name = rider, ap = 63.52%         (TP = 77, FP = 32)
class_id = 8, name = tempo, ap = 57.09%         (TP = 14, FP = 6)
class_id = 9, name = truck, ap = 71.33%         (TP = 81, FP = 21)

 for conf_thresh = 0.25, precision = 0.78, recall = 0.71, F1-score = 0.74
 for conf_thresh = 0.25, TP = 558, FP = 158, FN = 231, average IoU = 60.92 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.580831, or 58.08 %
```

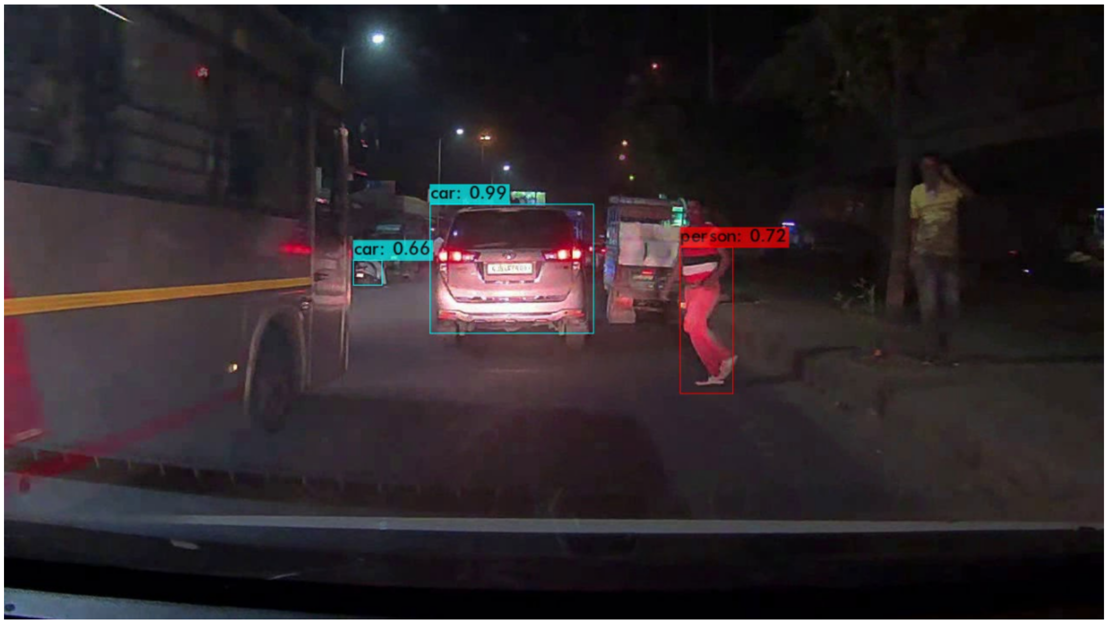
- The model achieved a **mAP of 58.08% and the loss is 0.33** (from the log file)
- Classes animal and bicycle are very poor in terms of precision. This is even lower than the YoloV4 precision for these classes due to less Convolution and Yolo layers.
- Only the class car has a decent precision due to over representation.

To get the inference speed, the same test image is used and the same GPU settings are applied. The YoloV4 - Tiny performs poorly on the image but the only good tradeoff is its inference speed.

The inference speed of YoloV4 tiny is **4.8ms** on a GPU:



```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 14.528
avg_outputs = 642675
 Allocate additional workspace_size = 26.22 MB
Loading weights from /content/drive/MyDrive/YOLOV4/backup_tiny2/yolov4-tiny-obj_19000.weights...
 seen 64, trained: 1216 K-images (19 Kilo-batches_64)
Done! Loaded 38 layers from weights-file
 Detection layer: 30 - type = 28
 Detection layer: 37 - type = 28
/content/drive/MyDrive/YOLOV4/video_rms_2020-11-07_18-54_128.jpg: Predicted in 4.898000 milli-seconds.
car: 66.017372%
car: 98.925949%
person: 72.157333%
```



The model is able to detect 3 objects in the sample image. For this model inference time is taken as the deciding factor. The aim is to detect objects within less time and little or no latency.

Here recall, F1 and avg IoU are for conf_thresh = 0.25:

Model	**Precision@0.5**	**Inference time**	**Recall**	**F1**	**Avg IoU**
YoloV4	62.03%	20ms	0.75	0.78	65.64%
YoloV4 - Tiny	58.08%	4.8ms	0.71	0,74	60.92%

Table 1: Model Result Summary

7. Observations

1. For the given dataset, the class distribution is not similar and data is heavily skewed and thus It is important to take a look at the mAP per class rather than the overall mAP.

2. For the 2 underrepresented classes, animal and bicycle, the mAP scores of YoloV4 tiny is very low with True Positive (TP) as 0 and 1 respectively. This precision is very high in the YoloV4 model. Considering the Recall, F1 scores and avg IoU at conf_threshold = 0.25, YoloV4 will be the final model used to provide predictions on the test images.

3. The spikes in the mAP curve in the charts is due to a less represented class coming in the validation set and reducing the overall performance. However, this is observed only in the earlier epochs, the curve smoothes out as the model converges.

4. The negative samples heavily affected the under represented class features and their accuracy. The model can accommodate the negative samples given that all the classes have equal distribution.

5. Setting *ignore_thresh=0.9* and *iou_loss=giou* decreased mAP at 0.5. This is not desirable as mAP at 0.5 gives the optimal bounding boxes given the dataset is not balanced.

Note that the YoloV4 tiny is just for experimenting and is primarily for edge devices and computers with low processing power. It is ideal for spaces where the memory is limited, as its weight file is approx 22MB compared to 244MB size of YoloV4 model weights.

To improve the accuracy further, the following steps can be adopted:

1. Dataset Optimisation:
    - Increase training data images.
    - Adding more samples for underrepresented classes to even out class distribution
    - Using images with high resolution so that there are more features to learn
    - Using separate training and validation dataset rather than splitting one big set. This will create a balanced class representation for both training and validation data.
    - Increase the variety of objects for the same class. The current dataset uses video stills and that creates a bias as the same object is repeated many times.
    - Augment data to add more features of the same object.
    - Using similar sized objects in both training and testing sets. That is, if only objects that occupied 80-90% of the image were present in the training set, then the trained network will not be able to detect objects that occupy 1-10% of the image.
    - If possible, remove objects that have less instances to improve accuracy.

2. Model Optimisation:
   - Increase the batch size and subdivisions. This gives more features to learn during a single epoch.
   - Increase the input image size. Higher resolutions give high precision.
   - Change the convolution filter size so that more features are extracted before every Yolo layer.
   - Recalculating anchor boxes for the image width and height, This is a tedious task but can greatly improve the accuracy.
   - Increase the image dimensions when in the detection stage, this can identify the small objects with higher accuracy.
   - For smaller sized images, increase the convolution stride so that features are not lost when it reaches the yolo layer.

8. Conclusion

The YoloV4 algorithm achieved a mAP of 62.03%. The loss at the end of training was 0.41. The model converged well though the data was highly skewed. It is possible to detect the objects in real time with this algorithm for the custom dataset objects. The model is able to detect objects with varied size and area values.

Submitted By:
Abhimanyu Saxena
Email: abhi.manyu1225@gmail.com
Phone: 7567877360