Objective: Identify cars and auto rickshaws in the traffic of Pune

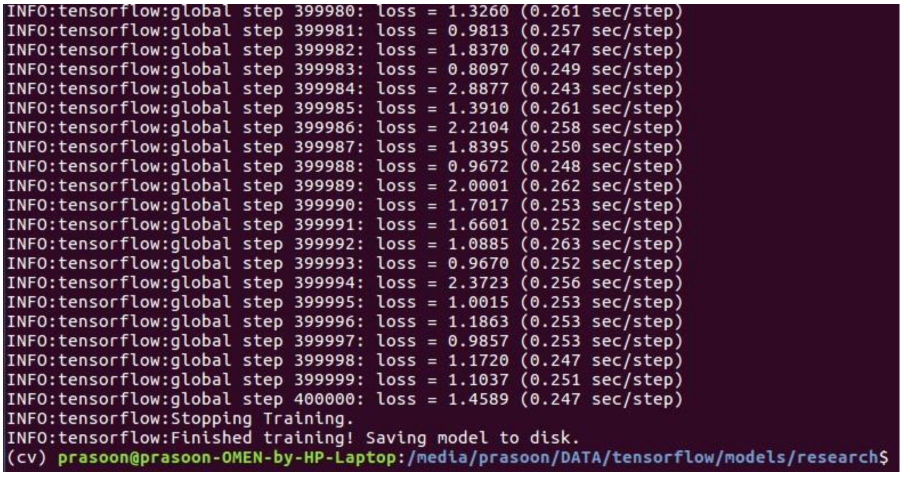
Prasoon Dhaneshwar

Engineer, Vijña Labs, Manipal Technologies Ltd.

M.Tech. Electronic Systems Design(Embedded Systems), IIIT Bangalore.

Challenges faced and arriving to conclusion

- Initial approach:
  - Used Darknet framework previously, to run the YOLO Network. Error convergence was very slow. And finally, the detection accuracy was not upto the mark(~70-80%).
- Used smaller network called SSD Moblienet V1, which is generally used in real time scenarios.
- TensorFlow gives a lot of freedom in terms of GPU utilisation, checkpoints when converging automatically.



```
INFO:tensorflow:global step 399980: loss = 1.3260 (0.261 sec/step)
INFO:tensorflow:global step 399981: loss = 0.9813 (0.257 sec/step)
INFO:tensorflow:global step 399982: loss = 1.8370 (0.247 sec/step)
INFO:tensorflow:global step 399983: loss = 0.8097 (0.249 sec/step)
INFO:tensorflow:global step 399984: loss = 2.8877 (0.243 sec/step)
INFO:tensorflow:global step 399985: loss = 1.3910 (0.261 sec/step)
INFO:tensorflow:global step 399986: loss = 2.2104 (0.258 sec/step)
INFO:tensorflow:global step 399987: loss = 1.8395 (0.250 sec/step)
INFO:tensorflow:global step 399988: loss = 0.9672 (0.248 sec/step)
INFO:tensorflow:global step 399989: loss = 2.0001 (0.262 sec/step)
INFO:tensorflow:global step 399990: loss = 1.7017 (0.253 sec/step)
INFO:tensorflow:global step 399991: loss = 1.6601 (0.252 sec/step)
INFO:tensorflow:global step 399992: loss = 1.0885 (0.263 sec/step)
INFO:tensorflow:global step 399993: loss = 0.9670 (0.252 sec/step)
INFO:tensorflow:global step 399994: loss = 2.3723 (0.256 sec/step)
INFO:tensorflow:global step 399995: loss = 1.0015 (0.253 sec/step)
INFO:tensorflow:global step 399996: loss = 1.1863 (0.253 sec/step)
INFO:tensorflow:global step 399997: loss = 0.9857 (0.253 sec/step)
INFO:tensorflow:global step 399998: loss = 1.1720 (0.247 sec/step)
INFO:tensorflow:global step 399999: loss = 1.1037 (0.251 sec/step)
INFO:tensorflow:global step 400000: loss = 1.4589 (0.247 sec/step)
INFO:tensorflow:Stopping Training.
INFO:tensorflow:Finished training! Saving model to disk.
(cv) prasoon@prasoon-OMEN-by-HP-Laptop:/media/prasoon/DATA/tensorflow/models/research$
```

Fig.1. Total 4,000,000 training steps in TensorFlow

CNN architecture, modifications, workflow

- SSD Moblienet V1 with 21 COCO Map for object detection.
- BBox-Label-Tool-multi-class for labelling.
- Convert the text annotation files to appropriate format needed by YOLO(for Darknet, initial approach).
- Stored image directories in train.txt and test.txt .
- Converted the annotations files and image details to tfrecord needed for training the pre-trained model.
- Edited SSD with Mobilenet v1 configuration for 2 classes, 4.000,000 steps, batch size of 8 with 344 images.
- Train the model.
- Freeze the graph after reaching avg loss rate closer to 1. Saving a Checkpoint Model (.ckpt) as a .pb File.
- Run the test script with the given video.
- Tune the threshold value to avoid false positives.

Resolution of the image for training and inference

- Most of the images were 960x720, and some downloaded from internet were 800x600.
- Converted the frames in the given video to images, for inference. Finally tested it in the given video.

GPU and CPU utilizations. How to reduce resource needs

- By reducing the batch size(total samples at any given time) while training, the GPU utilization can be lowered. Same methodology applies, when training on a CPU.
- Using smaller or relatively less complex networks like Moblienet which has a slight trade-off in accuracy, but very efficient for real time videos. The GPU will give more FPS in these scenarios.
- Lower resolution in video/image also reduces GPU utilization while testing the model. Same is true for lower FPS in a video.

```
prasoon@prasoon-OMEN-by-HP-Laptop:~$ vmstat 1 5
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
11  0      0 2732800 4666520 4324900    0    0    16    45  236    15 21  3 76  0  0
15  0      0 2711448 4666520 4324964    0    0     0     0 11395 39977 34  7 60  0  0
 2  0      0 2719912 4666520 4324948    0    0    16     0 11092 39379 35  5 58  2  0
 2  0      0 2738328 4666520 4324884    0    0     0     0 11271 41586 35  6 59  0  0
 1  0      0 2688384 4666520 4324852    0    0     0     0 11388 37277 34  6 60  0  0
```

Fig.2. CPU utilization

```
prasoon@prasoon-OMEN-by-HP-Laptop: ~
Every 0.1s: nvidia-smi                                   Thu Jul  5 23:06:59 2018

Thu Jul  5 23:06:59 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.26                 Driver Version: 396.26                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 105...  Off  | 00000000:01:00.0 Off |                  N/A |
| N/A   64C    P0    N/A /  N/A |   3525MiB /  4040MiB |     56%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1086      G   /usr/lib/xorg/Xorg                           184MiB |
|    0      2128      G   compiz                                       184MiB |
|    0     26960      C   python                                      3153MiB |
+-----------------------------------------------------------------------------+
```

Fig.3. GPU utilization

Image size impacting the results in accuracy and performance

- Resizing an image to lower value helps when the information required algorithm to learn from isn’t very complex for ex: dog vs cat. But, for complex objects like trees, reducing the image size reduces small scale detail on leaf shape in a way that's detrimental to the model. This affects the accuracy significantly. A typical rule of thumb is 800pixels by 600 pixels.
- When the objects to be classified differ very much, a smaller image will have sufficient feature vectors to recognize with a decent accuracy. Performance of the overall system in these situations increases.

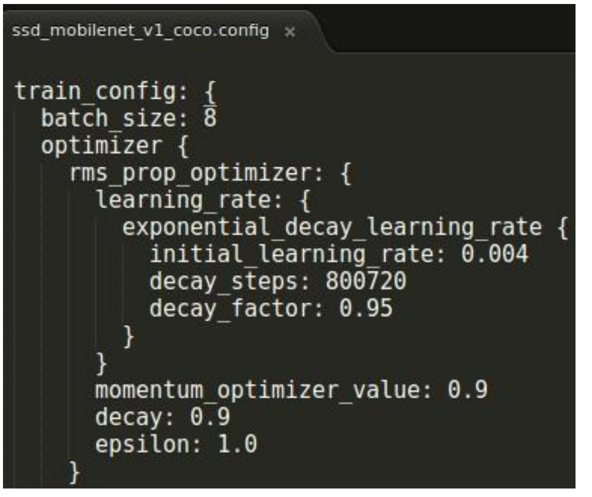


Fig.4. Configuring batch size

Metrics to measure accuracy

- The detection score written in the testing script, which is derived from the model. Refer line 72 and 76 in testing.py
- Uncomment line 77 in testing.py to measure fps. The speed of the video will be faster, since its running in GPU's multiple threads.
- Use Tensorboard from checkpoints in models/train/ to visually see the learning rate and convergence, no. of steps etc.
  - command: tensorboard --logdir path/to/logs

Usage, training and inference code

- Make sure you have tensorflow-gpu installed.
  - Follow these installation steps for dependencies and TensorFlow installation.
    - https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md
- Refer darknet_to_PascalVOC.py to convert Darknet labels to PASCAL VOC records.
- Create TFrecords
  - https://github.com/tensorflow/models/blob/master/research/object_detection/dataset_tools/create_pascal_tf_record.py
- Run the following command from tensorflow/models/research/ directory.
  - Refer train.py from link below in tensorflow/models/research/ directory
  - https://github.com/tensorflow/models/blob/master/research/object_detection/train.py
  - python object_detection/train.py \ --logtostderr \
    --pipeline_config_path=${PATH_TO_YOUR_PIPELINE_CONFIG} \
    --train_dir=${PATH_TO_TRAIN_DIR}
- Export the graph and save the checkpoint to .pb file:
  - python export_inference_graph.py --input_type image_tensor --pipeline_config_path ./rfcn_resnet101_coco.config --trained_checkpoint_prefix ./models/train/model.ckpt-5000 --output_directory ./fine_tuned_model
- Run the model:
  - Make sure you in OpenCV environment.
  - Run the following command from /tensorflow/models/research/object_detection/training_twoclass directory
    - python testing.py
    - Above script takes 'test.mp4' as a video stream and writes into 'output.mp4'