

Name: Amrita Sundari V

Email: amrita95@tamu.edu

Assignment

Machine Learning assessment - Report

Hardware information:

- Memory: 12 GB RAM
- Disk Memory: 62 GB
- Processor: Intel core i7-7500U CPU @2.70GHz x 4
- OS : Ubuntu 16.04 LTS
- Development Environment: PyCharm Edu 2017.3

GitHub repository: <https://github.com/amrita95/YOLO>

Section 1:

Github user y22ma modifies Darknet source code in order to train YOLO using Udacity's data. In this section, you should clone y22ma's project in your own GitHub repository, and then build the project on your machine.

I cloned the repository in my local directory from the attached link and checked out the branch, *udacity*. I chose one of the yolo architecture from the */cfg* folder and loaded the corresponding weights from the */bin* folder. The following command runs the algorithm on the test images.

```
./flow --model cfg/v1/yolo-full.cfg --load bin/yolo-full.weights --json False
```

The test images are stored in the */test* and the resulting output images with the bounding boxes gets stored in the */test/out*. In order to train on a dataset by loading a certain pre-trained weight, the command is

```
./flow --model cfg/v1/yolo-full.cfg --load bin/yolo-full.weights --dataset  
<dataset directory> --annotation <annotation file directory> --train
```

If we are to train the model from the scratch the `--load` has to be dropped from the above command. There were a few changes that were done since the version of tensorflow in my system was different from what is required. The syntax of few of the functions like `tf.concat`, `tf.multiply` had to be changed. The following are the changes that were made:

File Name	Line number	Changes
net/ops/convolution.py	88	key='param_initializers'
net/vanilla/train.py	27,28	Tf.mul -- > tf.multiply
net/yolo.py	62,68,72,85	Tf.concat - syntax change Tf.mul -- > tf.multiply

Section 2:

Train the network using Udacity's dataset.

To train on a new dataset, `--dataset` argument of `./flow` has to be set to the location of the udacity dataset. Y22ma's udacity branch of the repository has the udacity cfg file for 5 classes. The annotation file udacity.csv is parsed using a new function provided in the udacity branch `/utils/udacity_voc_csv.py`. The following command starts the training.

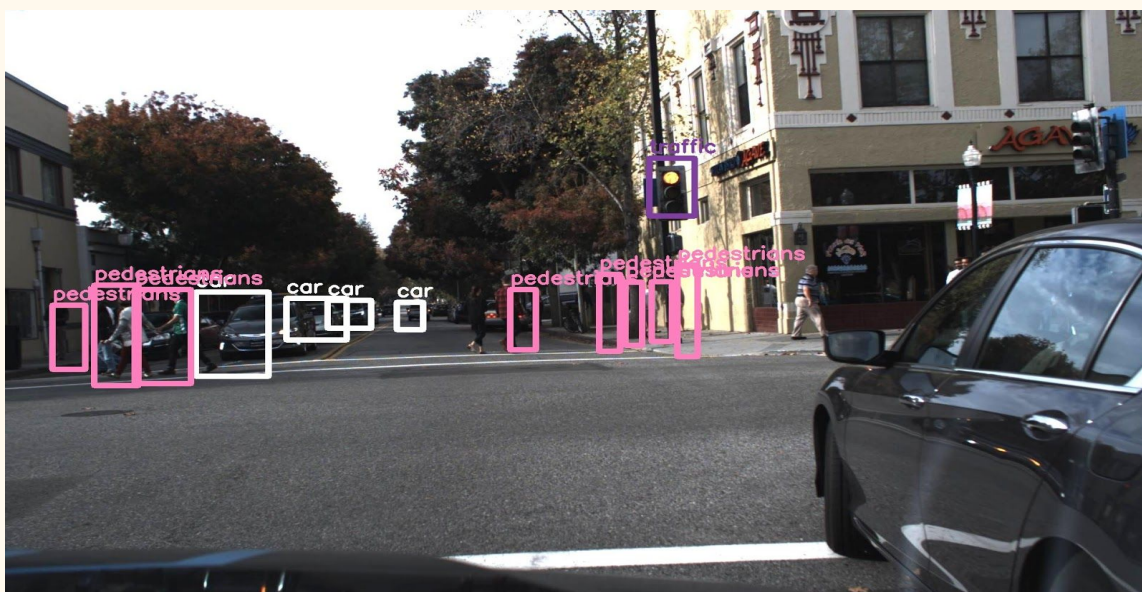
```
./flow --model cfg/tiny-yolo-udacity.cfg --load 9707 --dataset <dataset directory>
--annotation <csv file directory> --train
```

I loaded the weights from a checkpoint that was provided by y22ma and trained the network starting from the step 8987. To train from the beginning, I tried to overfit the model with a dataset of 5-6 images and the loss drastically reduced from ~100 to ~8. The loss doesn't reach 0 since there was a noise that was augmented to the images. This step is just to check if the whole system is working fine. But it has to run for at least 8000-9000 steps, before it can produce a meaningful bounding box.

The YOLO network model that I used is the one that was provided in the repository in the file `./cfg/tiny-yolo-udacity.cfg`. The architecture consisted of 9 convolutional layers, where each of the convolutional layer is followed by a max pooling layer except for the last 2 convolutional layers. The leaky ReLU is used as the activation function for all the layers except for the last layer where linear activation has been used. The screenshot attached below shows the architecture of the network in the `tiny-yolo-udacity.cfg`.

Building net ...					
Source	Train?	Layer description			Output size
		input			(?, 416, 416, 3)
Init	Yep!	conv	3x3p1_1	+bnorm leaky	(?, 416, 416, 16)
Load	Yep!	maxp	2x2p0_2		(?, 208, 208, 16)
Init	Yep!	conv	3x3p1_1	+bnorm leaky	(?, 208, 208, 32)
Load	Yep!	maxp	2x2p0_2		(?, 104, 104, 32)
Init	Yep!	conv	3x3p1_1	+bnorm leaky	(?, 104, 104, 64)
Load	Yep!	maxp	2x2p0_2		(?, 52, 52, 64)
Init	Yep!	conv	3x3p1_1	+bnorm leaky	(?, 52, 52, 128)
Load	Yep!	maxp	2x2p0_2		(?, 26, 26, 128)
Init	Yep!	conv	3x3p1_1	+bnorm leaky	(?, 26, 26, 256)
Load	Yep!	maxp	2x2p0_2		(?, 13, 13, 256)
Init	Yep!	conv	3x3p1_1	+bnorm leaky	(?, 13, 13, 512)
Load	Yep!	maxp	2x2p0_1		(?, 13, 13, 512)
Init	Yep!	conv	3x3p1_1	+bnorm leaky	(?, 13, 13, 1024)
Init	Yep!	conv	3x3p1_1	+bnorm leaky	(?, 13, 13, 1024)
Init	Yep!	conv	1x1p0_1	linear	(?, 13, 13, 50)

The training, if started with randomly initialized weights takes a lot of time and iterations (~8000) to converge. My PC ran out of memory with 600 steps and after 16 hours training, since I had saved the model for every 1000 data(800MB each). Training with pretrained weights (weights stored after ~9000 steps) gave a training loss ~8 in each steps later and moving average loss was also approximately around 9 . The screenshot below shows the training performance of the algorithm on udacity dataset loading the pre-trained weights. An example of the output image is shown below.



After ~9000 training steps, the bounding boxes are considerable and it can be improved even more by training it for more number of steps until the loss converges.

Section 3:

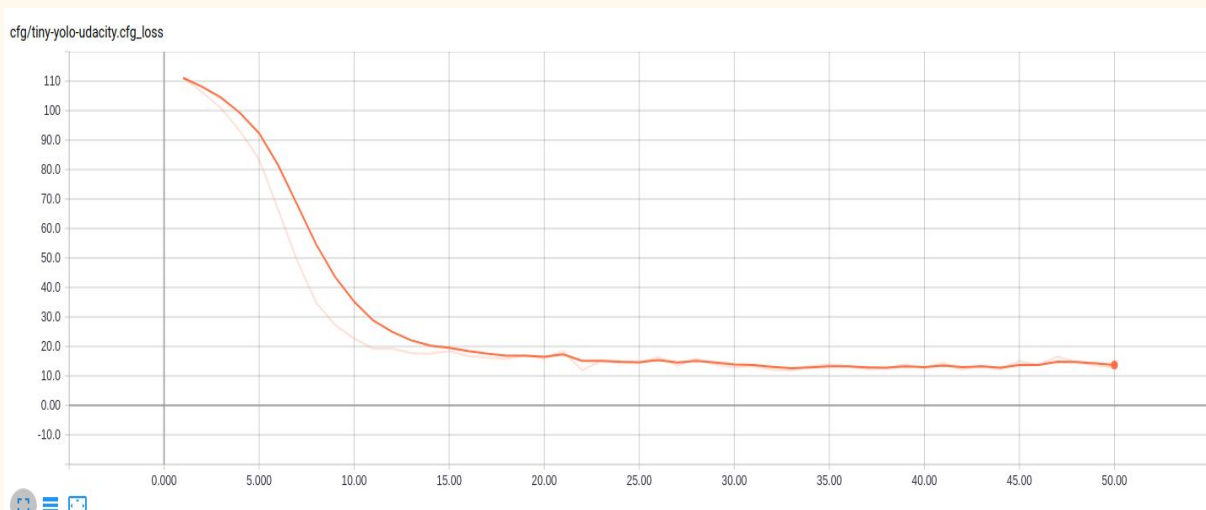
Provide more performance metrics. Tensorboard plotting is a good method to monitor the training. When running the detector, there are several ways to evaluate the performance.

TENSORBOARD:

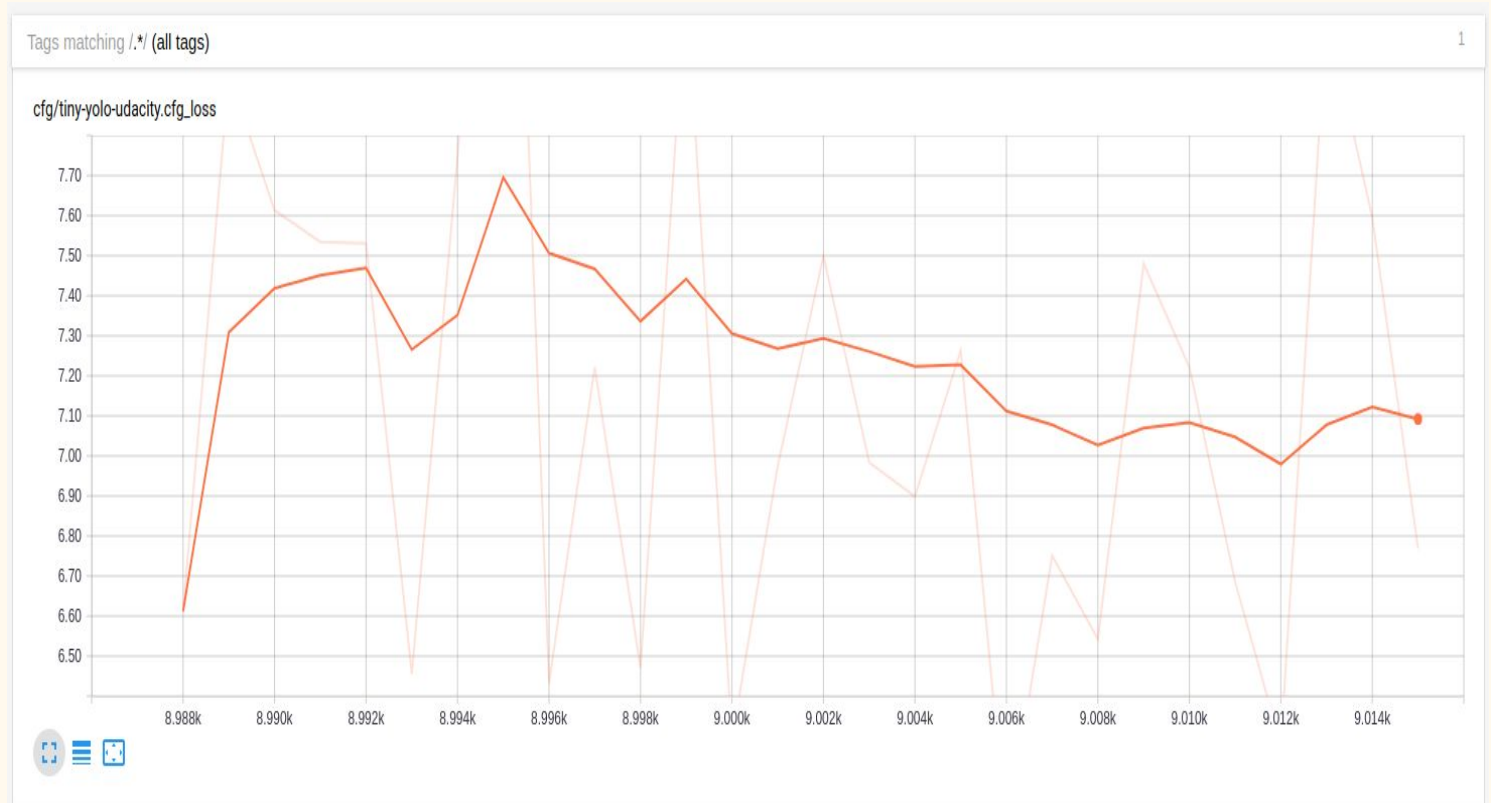
In order to add the tensorboard support to monitor the loss, I have modified the source code(/net/yolov2/train.py Line:107) by adding `tf.summary.scalar` ops to the node that gives loss as the output. `Tf.summary.merge_all()` will produce a serialized summary protobuf object and with the summary data at each step. Finally, to write it to the disk, I have passed the protobuf object to `tf.summary.FileWriter()`. These modifications can be seen at line 95 of the file `net/build.py`. I have also included a new argument `--summary` where you give the location to which the summary has to be stored. This can be called from the command line and the modification has been done in the file `/flow`. After storing, we can run the tensorboard using the following command.

```
tensorboard --logdir = <summary directory>
```

The following screenshot is the plot of the loss value plotted against the number of training steps, from the tensorboard. This loss value is actually from the model that was trained to overfit the small dataset(~5 images) in order to check if the network is working fine. The loss value has dropped from 112 to ~13 in just 50 iterations.



On the other hand, the plot of the loss value for a model loaded with pre trained weights, is oscillatory around the loss value of ~7.5 after 8900 iterations.



JSON:

I have modified the code `/net/yolov2/test.py` in order to output the JSON for each image and saved the JSON as a *text* file in the directory `/test/out`. I have also modified the file `/flow` in order to add a parameter `--json` of boolean type. When this parameter is set to true, the JSON output will be shown and it will be saved.

The following command will run the detector and will also save and the JSON is displayed as an output along with it. Here `--load 9707` denotes the loading of weights from a checkpoint saved at 9707th step.

```
./flow --model cfg/tiny-yolo-udacity.cfg --load 9707 --json True
```


The following screenshot shows the JSON output for each of image in the `/test` folder.

```
@amrita95-Inspiron-13-7378: ~/PycharmProjects/darkFlow2
Init | Yep! | conv 1x1p0_1 | linear | (?, 13, 13, 50)
-----
Running entirely on CPU
2018-03-23 03:10:24.390348: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use
Loading from ./ckpt/tiny-yolo-udacity-8987
Finished in 1.8278415203094482s

Forwarding 29 inputs ...
Total time = 4.242751359939575s / 29 inps = 6.8351872499106365 ips
Post processing 29 inputs ...
[Label: car, confidence: 0.52, topleft: {'x': 329, 'y': 539}, bottomright: {'x': 626, 'y': 720}}
[Label: car, confidence: 0.58, topleft: {'x': 919, 'y': 553}, bottomright: {'x': 1805, 'y': 614}}
[Label: car, confidence: 0.71, topleft: {'x': 815, 'y': 542}, bottomright: {'x': 902, 'y': 620}}
[Label: car, confidence: 0.64, topleft: {'x': 923, 'y': 582}, bottomright: {'x': 962, 'y': 630}}
[Label: car, confidence: 0.77, topleft: {'x': 890, 'y': 572}, bottomright: {'x': 943, 'y': 617}}
[Label: car, confidence: 0.51, topleft: {'x': 1408, 'y': 501}, bottomright: {'x': 1691, 'y': 681}}
[Label: car, confidence: 0.53, topleft: {'x': 1809, 'y': 546}, bottomright: {'x': 1896, 'y': 616}}
[Label: car, confidence: 0.65, topleft: {'x': 544, 'y': 554}, bottomright: {'x': 693, 'y': 632}}
[Label: car, confidence: 0.76, topleft: {'x': 275, 'y': 464}, bottomright: {'x': 877, 'y': 747}}
[Label: car, confidence: 0.51, topleft: {'x': 936, 'y': 557}, bottomright: {'x': 991, 'y': 601}}
[Label: car, confidence: 0.57, topleft: {'x': 591, 'y': 519}, bottomright: {'x': 783, 'y': 665}}
[Label: car, confidence: 0.57, topleft: {'x': 781, 'y': 548}, bottomright: {'x': 842, 'y': 609}}
[Label: car, confidence: 0.67, topleft: {'x': 1028, 'y': 548}, bottomright: {'x': 1124, 'y': 618}}
[Label: car, confidence: 0.5, topleft: {'x': 1500, 'y': 548}, bottomright: {'x': 1699, 'y': 630}}
[Label: car, confidence: 0.61, topleft: {'x': 645, 'y': 564}, bottomright: {'x': 755, 'y': 631}}
[Label: car, confidence: 0.52, topleft: {'x': 656, 'y': 596}, bottomright: {'x': 700, 'y': 652}}
[Label: car, confidence: 0.54, topleft: {'x': 540, 'y': 592}, bottomright: {'x': 613, 'y': 649}}
[Label: car, confidence: 0.55, topleft: {'x': 319, 'y': 575}, bottomright: {'x': 441, 'y': 747}}
[Label: car, confidence: 0.56, topleft: {'x': 471, 'y': 588}, bottomright: {'x': 572, 'y': 673}}
[Label: pedestrians, confidence: 0.6, topleft: {'x': 149, 'y': 561}, bottomright: {'x': 223, 'y': 766}}
[Label: pedestrians, confidence: 0.6, topleft: {'x': 79, 'y': 602}, bottomright: {'x': 134, 'y': 733}}
[Label: pedestrians, confidence: 0.62, topleft: {'x': 1039, 'y': 555}, bottomright: {'x': 1067, 'y': 684}}
[Label: pedestrians, confidence: 0.65, topleft: {'x': 1083, 'y': 554}, bottomright: {'x': 1117, 'y': 676}}
[Label: pedestrians, confidence: 0.66, topleft: {'x': 217, 'y': 570}, bottomright: {'x': 311, 'y': 762}}
[Label: pedestrians, confidence: 0.7, topleft: {'x': 846, 'y': 570}, bottomright: {'x': 889, 'y': 693}}
[Label: pedestrians, confidence: 0.7, topleft: {'x': 995, 'y': 539}, bottomright: {'x': 1030, 'y': 696}}
[Label: pedestrians, confidence: 0.72, topleft: {'x': 1126, 'y': 520}, bottomright: {'x': 1160, 'y': 707}}
[Label: traffic, confidence: 0.6, topleft: {'x': 1079, 'y': 302}, bottomright: {'x': 1155, 'y': 420}}
[Label: car, confidence: 0.65, topleft: {'x': 486, 'y': 565}, bottomright: {'x': 657, 'y': 676}}
[Label: car, confidence: 0.67, topleft: {'x': 616, 'y': 554}, bottomright: {'x': 805, 'y': 679}}
[Label: car, confidence: 0.73, topleft: {'x': 892, 'y': 565}, bottomright: {'x': 980, 'y': 620}}
[Label: car, confidence: 0.51, topleft: {'x': 1034, 'y': 544}, bottomright: {'x': 1152, 'y': 632}}
[Label: car, confidence: 0.51, topleft: {'x': 1474, 'y': 457}, bottomright: {'x': 1881, 'y': 716}}
[Label: car, confidence: 0.72, topleft: {'x': 397, 'y': 551}, bottomright: {'x': 592, 'y': 662}}
[Label: car, confidence: 0.59, topleft: {'x': 953, 'y': 545}, bottomright: {'x': 1055, 'y': 609}}
[Label: car, confidence: 0.59, topleft: {'x': 575, 'y': 543}, bottomright: {'x': 684, 'y': 617}}
[Label: car, confidence: 0.54, topleft: {'x': 870, 'y': 550}, bottomright: {'x': 963, 'y': 612}}
[Label: car, confidence: 0.54, topleft: {'x': 938, 'y': 554}, bottomright: {'x': 1031, 'y': 610}}
[Label: car, confidence: 0.66, topleft: {'x': 608, 'y': 567}, bottomright: {'x': 684, 'y': 631}}
[Label: car, confidence: 0.51, topleft: {'x': 648, 'y': 576}, bottomright: {'x': 747, 'y': 646}}
[Label: car, confidence: 0.51, topleft: {'x': 760, 'y': 565}, bottomright: {'x': 825, 'y': 615}}
[Label: car, confidence: 0.53, topleft: {'x': 724, 'y': 559}, bottomright: {'x': 790, 'y': 617}}
```

Additional Key Performance Indicators (KPI):

I have modified the `/net/yolov2/test.py` file and also added a new file in the `utils` directory `/utils/additionalKPI.py`. The new file contains the code, which calculates the number of actual bounding boxes in each image (Total), the number of bounding boxes predicted by the network (Proposal) and also the number of predicted boxes which are correct by thresholding on the IOU of the predicted and actual bounding box(Correct). Now the recall and precision for each of the image can be found as correct/total and correct/predicted respectively. The average IOU is average of all the IOUs that are taken between predicted and actual bounding boxes in a single image. I have also added a boolean feature in `/flow` to output these values whenever that parameter is set true. The command to display the KPIs is as follows.

```
./flow --model cfg/tiny-yolo-udacity.cfg --load 9707 --json False --kpi True
```


The screenshot below shows the new KPIs for the test images.

```
@amrita95-Inspiron-13-7378: ~/PycharmProjects/darkflow2
Load | Yep! | maxp 2x2p0_2 | (? , 104, 104, 32)
Init | Yep! | conv 3x3p1_1 | (? , 104, 104, 64)
Load | Yep! | maxp 2x2p0_2 | (? , 52, 52, 64)
Init | Yep! | conv 3x3p1_1 | (? , 52, 52, 128)
Load | Yep! | maxp 2x2p0_2 | (? , 26, 26, 128)
Init | Yep! | conv 3x3p1_1 | (? , 26, 26, 256)
Load | Yep! | maxp 2x2p0_2 | (? , 13, 13, 256)
Init | Yep! | conv 3x3p1_1 | (? , 13, 13, 512)
Load | Yep! | maxp 2x2p0_1 | (? , 13, 13, 512)
Init | Yep! | conv 3x3p1_1 | (? , 13, 13, 1024)
Init | Yep! | conv 3x3p1_1 | (? , 13, 13, 1024)
Init | Yep! | conv 1x1p0_1 | linear | (? , 13, 13, 50)
-----
unning entirely on CPU
018-03-23 03:08:53.176113: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
loading from ./ckpt/tiny-yolo-udacity-8987
inished in 1.8411083221435547s

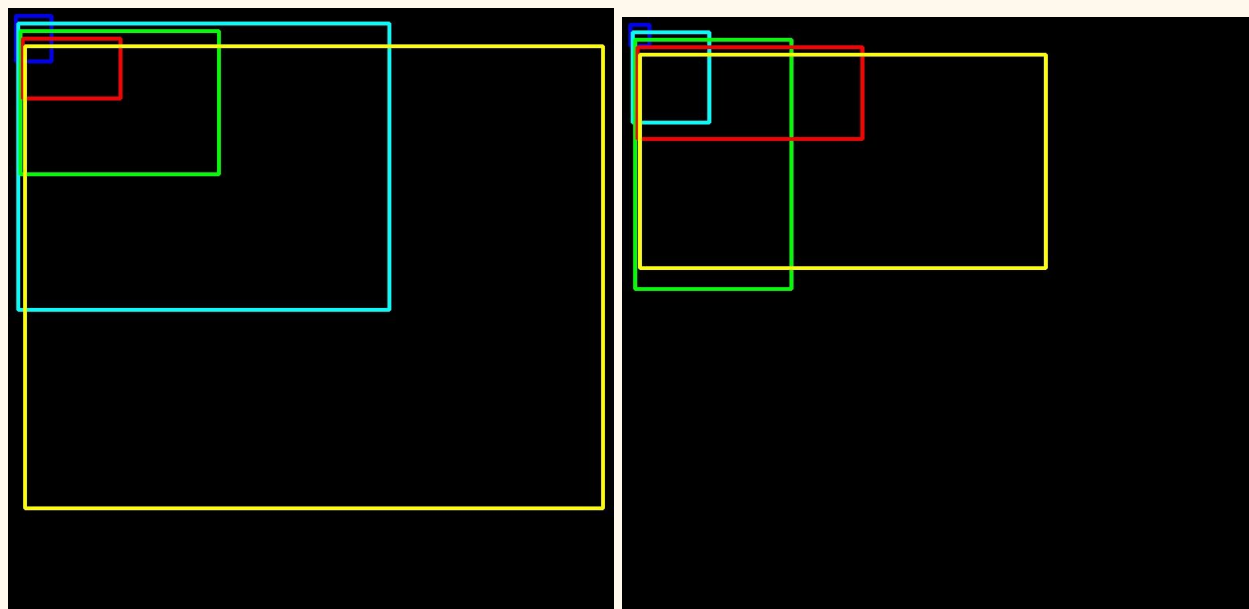
forwarding 29 inputs ...
total time = 4.090061187744141s / 29 inps = 7.090358473584316 ips
ost processing 29 inputs ...
mg: 1478021859580454463.jpg Total: 2 Proposals: 2 Correct: 1 AvgIOU: 0.204536 Recall: 0.500000 Precision: 0.500000
mg: 1478021864080939917.jpg Total: 2 Proposals: 1 Correct: 1 AvgIOU: 0.397935 Recall: 0.500000 Precision: 1.000000
mg: 1478021860587172205.jpg Total: 5 Proposals: 2 Correct: 1 AvgIOU: 0.152173 Recall: 0.200000 Precision: 0.500000
mg: 1478021855087664141.jpg Total: 4 Proposals: 3 Correct: 3 AvgIOU: 0.460172 Recall: 0.750000 Precision: 1.000000
mg: 1478021866580378914.jpg Total: 2 Proposals: 1 Correct: 1 AvgIOU: 0.198091 Recall: 0.500000 Precision: 1.000000
mg: 1478021866080399860.jpg Total: 2 Proposals: 2 Correct: 1 AvgIOU: 0.376533 Recall: 0.500000 Precision: 0.500000
mg: 1478021867580627987.jpg Total: 2 Proposals: 2 Correct: 1 AvgIOU: 0.269537 Recall: 0.500000 Precision: 0.500000
mg: 1478021854080438284.jpg Total: 4 Proposals: 2 Correct: 2 AvgIOU: 0.271525 Recall: 0.500000 Precision: 1.000000
mg: 1478899614569509669.jpg Total: 21 Proposals: 13 Correct: 8 AvgIOU: 0.195322 Recall: 0.380952 Precision: 0.615385
mg: 1478021859086040035.jpg Total: 1 Proposals: 2 Correct: 1 AvgIOU: 0.664074 Recall: 1.000000 Precision: 0.500000
mg: 1478021862080365415.jpg Total: 8 Proposals: 1 Correct: 1 AvgIOU: 0.042504 Recall: 0.125000 Precision: 1.000000
mg: 1478021855579897603.jpg Total: 5 Proposals: 3 Correct: 3 AvgIOU: 0.373319 Recall: 0.600000 Precision: 1.000000
mg: 1478021856580199398.jpg Total: 2 Proposals: 2 Correct: 2 AvgIOU: 0.475551 Recall: 1.000000 Precision: 1.000000
mg: 1478021857586940196.jpg Total: 3 Proposals: 3 Correct: 3 AvgIOU: 0.455850 Recall: 1.000000 Precision: 1.000000
mg: 1478021853580207224.jpg Total: 4 Proposals: 4 Correct: 3 AvgIOU: 0.358317 Recall: 0.750000 Precision: 0.750000
mg: 1478021857080179151.jpg Total: 3 Proposals: 2 Correct: 2 AvgIOU: 0.480211 Recall: 0.666667 Precision: 1.000000
mg: 1478021854586499411.jpg Total: 4 Proposals: 3 Correct: 2 AvgIOU: 0.244573 Recall: 0.500000 Precision: 0.666667
mg: 1478021863080437795.jpg Total: 8 Proposals: 2 Correct: 1 AvgIOU: 0.097566 Recall: 0.125000 Precision: 0.500000
mg: 1478021853086493872.jpg Total: 5 Proposals: 3 Correct: 2 AvgIOU: 0.298649 Recall: 0.400000 Precision: 0.666667
mg: 1478021864589356545.jpg Total: 2 Proposals: 1 Correct: 1 AvgIOU: 0.284043 Recall: 0.500000 Precision: 1.000000
mg: 1478021865088845201.jpg Total: 2 Proposals: 2 Correct: 1 AvgIOU: 0.175077 Recall: 0.500000 Precision: 0.500000
mg: 1478021865581271045.jpg Total: 2 Proposals: 2 Correct: 1 AvgIOU: 0.326693 Recall: 0.500000 Precision: 0.500000
mg: 1478021861080275540.jpg Total: 5 Proposals: 4 Correct: 2 AvgIOU: 0.289417 Recall: 0.400000 Precision: 0.500000
mg: 1478021856081580734.jpg Total: 3 Proposals: 5 Correct: 3 AvgIOU: 0.634126 Recall: 1.000000 Precision: 0.600000
mg: 1478021863580052463.jpg Total: 3 Proposals: 3 Correct: 2 AvgIOU: 0.506803 Recall: 0.666667 Precision: 0.666667
mg: 1478021860086872697.jpg Total: 4 Proposals: 3 Correct: 2 AvgIOU: 0.395856 Recall: 0.500000 Precision: 0.666667
mg: 1478021861579942332.jpg Total: 4 Proposals: 4 Correct: 1 AvgIOU: 0.253268 Recall: 0.250000 Precision: 0.250000
mg: 1478021862586653474.jpg Total: 8 Proposals: 1 Correct: 0 AvgIOU: 0.000000 Recall: 0.000000 Precision: 0.000000
mg: 1478021867087816086.jpg Total: 1 Proposals: 3 Correct: 1 AvgIOU: 0.822295 Recall: 1.000000 Precision: 0.333333
total time = 2.9229414463043213s / 29 inps = 9.92151246706181 ips
mrta95@amrita95-Inspiron-13-7378:~/PycharmProjects/darkFlow2$
```

Section 4:

YOLO uses pre-calculated anchor box in the training. The original YOLO's anchor boxes are calculated from VOC 2007 dataset using K-mean cluster algorithm.

To find the anchor boxes, I have first extracted the width and height of all the bounding boxes present in the images of udacity dataset and applied K-means algorithm using width and height as the feature. I took the number of clusters to be 5, one for each class in the udacity dataset. I have included a new file for finding the anchor boxes (`/utils/kmeans_anchor.py`). The following is a representation of the anchor boxes found using k-means. Based on

intuition, as we can see the anchor boxes derived from VOC dataset would perform better than the udacity anchor boxes.



Udacity Anchor boxes

VOC dataset anchor boxes

After fitting the K-means model on the udacity dataset, the centre points of each of the clusters are assumed to be the width and height of the anchor boxes. These were then converted to the scale of 13 x 13 in order to be provided to the network. The anchor box data is provided in the table below.

	<i>Width</i>	<i>Height</i>
<i>Anchor box 1</i>	<i>1.79455642</i>	<i>2.19526107</i>
<i>Anchor box 2</i>	<i>8.7824603</i>	<i>6.90141213</i>
<i>Anchor box 3</i>	<i>24.54700779</i>	<i>20.62964766</i>
<i>Anchor box 4</i>	<i>15.78625389</i>	<i>12.48136682</i>
<i>Anchor box 5</i>	<i>4.68438428</i>	<i>3.74576042</i>

The performance of the detector using the new anchor boxes is shown below in the screenshot. The output produced by the network using the new anchor boxes is shown in the second image below.

```
amrita95@amrita95-Inspiron-13-7378: ~/PycharmProjects/darkflow2
Load   Yep!   maxp  2x2p0_2   (? , 104, 104, 32)
Init   Yep!   conv  3x3p1_1   +bnorm leaky   (? , 104, 104, 64)
Load   Yep!   maxp  2x2p0_2   (? , 52, 52, 64)
Init   Yep!   conv  3x3p1_1   +bnorm leaky   (? , 52, 52, 128)
Load   Yep!   maxp  2x2p0_2   (? , 26, 26, 128)
Init   Yep!   conv  3x3p1_1   +bnorm leaky   (? , 26, 26, 256)
Load   Yep!   maxp  2x2p0_2   (? , 13, 13, 256)
Init   Yep!   conv  3x3p1_1   +bnorm leaky   (? , 13, 13, 512)
Load   Yep!   maxp  2x2p0_1   (? , 13, 13, 512)
Init   Yep!   conv  3x3p1_1   +bnorm leaky   (? , 13, 13, 1024)
Init   Yep!   conv  3x3p1_1   +bnorm leaky   (? , 13, 13, 1024)
Init   Yep!   conv  1x1p0_1   llinear        (? , 13, 13, 50)

-----
Running entirely on CPU
2018-03-23 19:47:32.312492: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use:
Loading from ./ckpt/tiny-yolo-udacity-anchr-8987
Finished in 1.962388277053833s

Forwarding 29 inputs ...
Total time = 3.759228229522705s / 29 inps = 7.714349390189065 ips
Post processing 29 inputs ...
Total: 2      Proposals: 3      Correct: 1      AvgIOU: 0.315675      Recall: 0.500000      Precision: 0.333333
Img: 1478021859580454463.jpg      Total: 2      Proposals: 2      Correct: 1      AvgIOU: 0.225680      Recall: 0.500000      Precision: 0.500000
Img: 1478021864080939017.jpg      Total: 2      Proposals: 2      Correct: 1      AvgIOU: 0.090593      Recall: 0.200000      Precision: 0.500000
Img: 1478021860587172205.jpg      Total: 5      Proposals: 4      Correct: 2      AvgIOU: 0.204120      Recall: 0.500000      Precision: 0.500000
Img: 1478021855987664141.jpg      Total: 4      Proposals: 2      Correct: 0      AvgIOU: 0.264700      Recall: 0.000000      Precision: 0.000000
Img: 1478021866580378914.jpg      Total: 2      Proposals: 3      Correct: 1      AvgIOU: 0.430863      Recall: 0.500000      Precision: 0.333333
Img: 1478021866080399860.jpg      Total: 2      Proposals: 3      Correct: 0      AvgIOU: 0.235400      Recall: 0.000000      Precision: 0.000000
Img: 1478021867580627987.jpg      Total: 2      Proposals: 4      Correct: 2      AvgIOU: 0.247876      Recall: 0.500000      Precision: 0.500000
Img: 1478021854080438284.jpg      Total: 4      Proposals: 4      Correct: 2      AvgIOU: 0.115300      Recall: 0.095238      Precision: 0.153846
Img: 147899614569589669.jpg      Total: 21      Proposals: 13      Correct: 2      AvgIOU: 0.473086      Recall: 1.000000      Precision: 0.333333
Img: 1478021859086040035.jpg      Total: 1      Proposals: 3      Correct: 1      AvgIOU: 0.473086      Recall: 1.000000      Precision: 0.333333
Img: 1478021862080365415.jpg      Total: 8      Proposals: 2      Correct: 0      AvgIOU: 0.038014      Recall: 0.000000      Precision: 0.000000
Img: 1478021855579897603.jpg      Total: 5      Proposals: 5      Correct: 1      AvgIOU: 0.199913      Recall: 0.200000      Precision: 0.200000
Img: 1478021856580199398.jpg      Total: 2      Proposals: 4      Correct: 1      AvgIOU: 0.508046      Recall: 0.500000      Precision: 0.250000
Img: 1478021857580940196.jpg      Total: 3      Proposals: 4      Correct: 1      AvgIOU: 0.337282      Recall: 0.333333      Precision: 0.250000
Img: 1478021853580207224.jpg      Total: 4      Proposals: 4      Correct: 2      AvgIOU: 0.271471      Recall: 0.500000      Precision: 0.500000
Img: 1478021857080179151.jpg      Total: 3      Proposals: 4      Correct: 1      AvgIOU: 0.330809      Recall: 0.333333      Precision: 0.250000
Img: 1478021854586499411.jpg      Total: 4      Proposals: 5      Correct: 1      AvgIOU: 0.283357      Recall: 0.250000      Precision: 0.200000
Img: 1478021863080437795.jpg      Total: 8      Proposals: 3      Correct: 1      AvgIOU: 0.042265      Recall: 0.125000      Precision: 0.333333
Img: 1478021853086493872.jpg      Total: 5      Proposals: 4      Correct: 1      AvgIOU: 0.165943      Recall: 0.200000      Precision: 0.250000
Img: 1478021864589356545.jpg      Total: 2      Proposals: 2      Correct: 1      AvgIOU: 0.263546      Recall: 0.500000      Precision: 0.500000
Img: 1478021865088845201.jpg      Total: 2      Proposals: 3      Correct: 1      AvgIOU: 0.269990      Recall: 0.500000      Precision: 0.333333
Img: 1478021865581271045.jpg      Total: 2      Proposals: 3      Correct: 1      AvgIOU: 0.256260      Recall: 0.500000      Precision: 0.333333
Img: 1478021861080275540.jpg      Total: 5      Proposals: 3      Correct: 1      AvgIOU: 0.107540      Recall: 0.200000      Precision: 0.333333
Img: 1478021856081580734.jpg      Total: 3      Proposals: 7      Correct: 2      AvgIOU: 0.424500      Recall: 0.666667      Precision: 0.285714
Img: 1478021863580052463.jpg      Total: 3      Proposals: 4      Correct: 2      AvgIOU: 0.255273      Recall: 0.666667      Precision: 0.500000
Img: 1478021860086872697.jpg      Total: 4      Proposals: 4      Correct: 1      AvgIOU: 0.187312      Recall: 0.250000      Precision: 0.250000
Img: 1478021861579942332.jpg      Total: 4      Proposals: 3      Correct: 1      AvgIOU: 0.164226      Recall: 0.250000      Precision: 0.333333
Img: 1478021862586653474.jpg      Total: 8      Proposals: 1      Correct: 0      AvgIOU: 0.000000      Recall: 0.000000      Precision: 0.000000
Img: 1478021867087816086.jpg      Total: 1      Proposals: 3      Correct: 0      AvgIOU: 0.216924      Recall: 0.000000      Precision: 0.000000
Total time = 2.8425841331481934s / 29 inps = 10.20198475810184 ips
amrita95@amrita95-Inspiron-13-7378: ~/PycharmProjects/darkflow2$
```



Section 5:

Can you improve the network to get a better result?

1. To improve the object detection, I tried increasing the resolution of the image. This improved the precision of the detection, and the network was able to predict smaller objects with more precision. I implemented this as a modification in the cfg file ./cfg/tiny-yolo-udacity.cfg by changing the values of height and width from 416 to 608 or 832 (or any multiples of 32). The KPIs after the change is shown in the screenshot below.

```
Forwarding 29 inputs ...
Total time = 17.915765047073364s / 29 inps = 1.6186861082294282 ips
Post processing 29 inputs ...
Img: 1478021859580454463.jpg Total: 2 Proposals: 5 Correct: 1 AvgIOU: 0.178796 Recall: 0.500000 Precision: 0.200000
Img: 1478021864080939917.jpg Total: 2 Proposals: 2 Correct: 2 AvgIOU: 0.800348 Recall: 1.000000 Precision: 1.000000
Img: 1478021860587172205.jpg Total: 5 Proposals: 3 Correct: 0 AvgIOU: 0.115062 Recall: 0.000000 Precision: 0.000000
Img: 1478021855087664141.jpg Total: 4 Proposals: 3 Correct: 1 AvgIOU: 0.128645 Recall: 0.250000 Precision: 0.333333
Img: 1478021866580378914.jpg Total: 2 Proposals: 6 Correct: 2 AvgIOU: 0.660984 Recall: 1.000000 Precision: 0.333333
Img: 1478021866080399860.jpg Total: 2 Proposals: 4 Correct: 2 AvgIOU: 0.774210 Recall: 1.000000 Precision: 0.500000
Img: 1478021867580627987.jpg Total: 2 Proposals: 3 Correct: 2 AvgIOU: 0.465384 Recall: 1.000000 Precision: 0.666667
Img: 1478021854080438284.jpg Total: 4 Proposals: 4 Correct: 2 AvgIOU: 0.353844 Recall: 0.500000 Precision: 0.500000
Img: 1478899614569509669.jpg Total: 21 Proposals: 7 Correct: 4 AvgIOU: 0.091149 Recall: 0.190476 Precision: 0.571429
Img: 1478021859086040035.jpg Total: 1 Proposals: 6 Correct: 1 AvgIOU: 0.692903 Recall: 1.000000 Precision: 0.166667
Img: 1478021862080365415.jpg Total: 8 Proposals: 3 Correct: 2 AvgIOU: 0.152844 Recall: 0.250000 Precision: 0.666667
Img: 1478021855579897603.jpg Total: 5 Proposals: 4 Correct: 3 AvgIOU: 0.363031 Recall: 0.600000 Precision: 0.750000
Img: 1478021856580199398.jpg Total: 2 Proposals: 4 Correct: 2 AvgIOU: 0.532471 Recall: 1.000000 Precision: 0.500000
Img: 1478021857586940196.jpg Total: 3 Proposals: 2 Correct: 2 AvgIOU: 0.333627 Recall: 0.666667 Precision: 1.000000
Img: 1478021853580207224.jpg Total: 4 Proposals: 7 Correct: 2 AvgIOU: 0.299466 Recall: 0.500000 Precision: 0.285714
Img: 1478021857080179151.jpg Total: 3 Proposals: 3 Correct: 3 AvgIOU: 0.629527 Recall: 1.000000 Precision: 1.000000
Img: 1478021854586499411.jpg Total: 4 Proposals: 3 Correct: 2 AvgIOU: 0.268172 Recall: 0.500000 Precision: 0.666667
Img: 1478021863080437795.jpg Total: 8 Proposals: 3 Correct: 2 AvgIOU: 0.101607 Recall: 0.250000 Precision: 0.666667
Img: 1478021853086493872.jpg Total: 5 Proposals: 4 Correct: 1 AvgIOU: 0.146627 Recall: 0.200000 Precision: 0.250000
Img: 1478021864589356545.jpg Total: 2 Proposals: 3 Correct: 2 AvgIOU: 0.450589 Recall: 1.000000 Precision: 0.666667
Img: 1478021865088845201.jpg Total: 2 Proposals: 3 Correct: 2 AvgIOU: 0.566301 Recall: 1.000000 Precision: 0.666667
Img: 1478021865581271045.jpg Total: 2 Proposals: 5 Correct: 2 AvgIOU: 0.743762 Recall: 1.000000 Precision: 0.400000
Img: 1478021861080275540.jpg Total: 5 Proposals: 5 Correct: 1 AvgIOU: 0.273671 Recall: 0.200000 Precision: 0.200000
Img: 1478021856081580734.jpg Total: 3 Proposals: 2 Correct: 2 AvgIOU: 0.437341 Recall: 0.666667 Precision: 1.000000
Img: 1478021863580052463.jpg Total: 3 Proposals: 3 Correct: 3 AvgIOU: 0.578902 Recall: 1.000000 Precision: 1.000000
Img: 1478021860086872697.jpg Total: 4 Proposals: 3 Correct: 1 AvgIOU: 0.212242 Recall: 0.250000 Precision: 0.333333
Img: 1478021861579942332.jpg Total: 4 Proposals: 4 Correct: 1 AvgIOU: 0.221056 Recall: 0.250000 Precision: 0.250000
Img: 1478021862586653474.jpg Total: 8 Proposals: 5 Correct: 4 AvgIOU: 0.250552 Recall: 0.500000 Precision: 0.800000
Img: 1478021867087816086.jpg Total: 1 Proposals: 5 Correct: 1 AvgIOU: 0.592133 Recall: 1.000000 Precision: 0.200000
Total time = 6.419684410095215s / 29 inps = 4.51735601743854 ips
```

The output of one image is shown below. As we can see, the network was able to predict even a car that was very far away with high precision.



2. Other suggestions to improve the performance (which I haven't tried out) are:

- Increase the resolution of the network before training of the model. This would possibly improve the precision but the training consumed more time than expected. (ran only 400 iterations in 12 hours in my laptop). This improvement is again related to changing the height and width of the input in the `.cfg` file that we are using.
- Augment more data in the training dataset, with different scales, rotation, lighting setup etc. Also, augment data with no objects of concern (negative examples) where you want your network to predict nothing.