Lyft Perception Challenge Writeup

Ong Chin Kiat

Abstract—A Fully Convolutional Network (FCN) is trained to identify road surface and other vehicles from images recorded using a forward facing camera on a car. Using a FCN network initially designed for a school assignment as a base, the author successfully transform it into a high performance semantic segmentation engine that performed well enough to stay in the top region of the Leaderboard, at least for the first half of the challenge.

Index Terms—Udacity, Lyft, Self Driving Car, FCN

1 Introduction

For this challenge, I built and trained a Fully Convolutional Network (FCN) using Keras. The Network is trained to identify road surface and other vehicles from images recorded using a forward facing camera on a car.

2 Data Set

For this challenge, the images were captured using the CARLA simulator. Images were captured using these camera setting:

```
camera2 = Camera('CameraRGB')
camera2.set_image_size(800, 600)
camera2.set_position(1.30, 0, 1.30)
camera2.set(FOV=90.0)
settings.add_sensor(camera2)
```

An initial data set of 1000 images were provided by Udacity. We are encouraged to do our own data capture using the CARLA simulator.

The dataset consists of images (Figure 1) and the corresponding ground truth pixel-wise labels (Figure 2) for each image.

Fig. 1. Example Image Captured

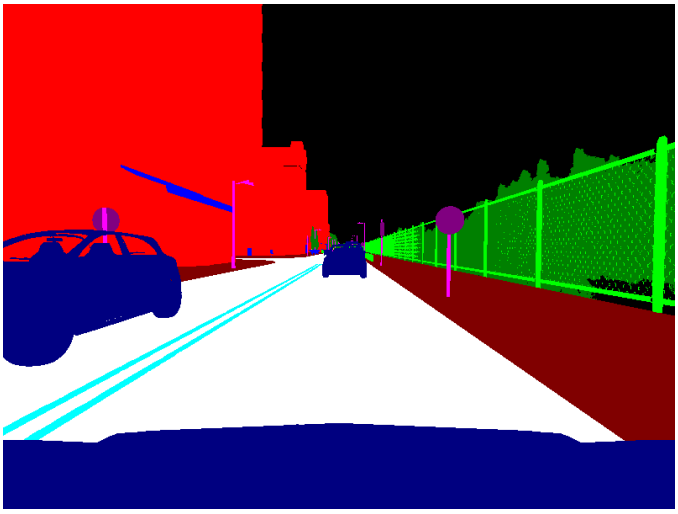


Fig. 2. Example Image Ground Truth

Since we are only required to identify the road surface and other vehicles, the first step is to write a simple script to process the given data set.

For the car label, we need to remove our own car hood from the label result. I take in the pixels with label 10, and remove all from row 496 to row 600.

For the road label, we combine the road label (7) and the road marking label (6).

I mapped the car labels to the green channel, road labels to the blue channels, and set all the other pixels to red. The resulting ground truth image is shown in Figure 3.

This looks very similar to the Robotics Nanodegree Project "Follow Me" which I did 8 months ago. In that project, we were using images captured from a Quad-Copter moving in a Quad-Copter Simulator. In Figure 4, the left image is the camera image captured, the centre is the ground truth data (Blue: Target Hero, Green: other people, Red: background), and the right image is the prediction done by the trained FCN.

Thus the first thing I try is to dig out the Jupyter Notebook for that project, and start adopting it to our CARLA data set.

The following Network Architecture section is copied from my own project submission report for the "Follow

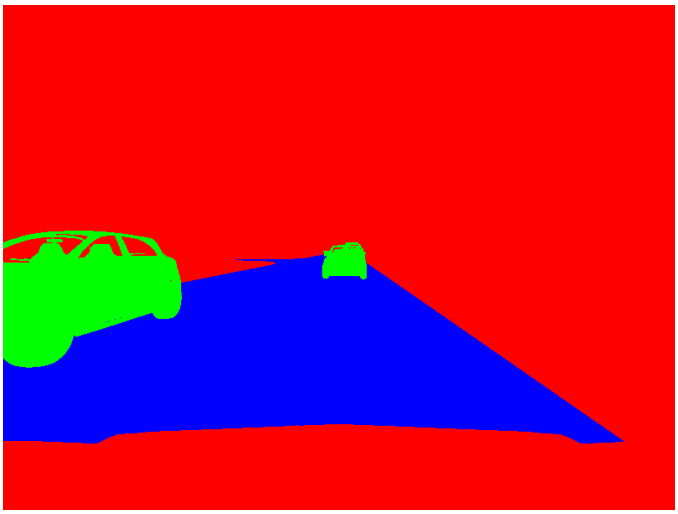


Fig. 3. Example Segmentation Image

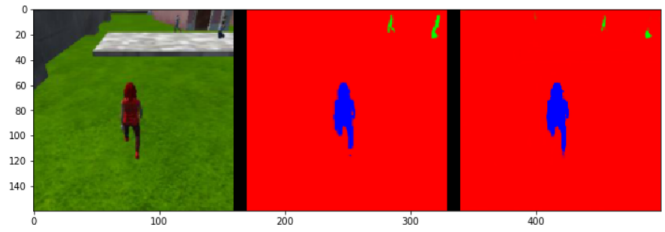


Fig. 4. Robotics Nanodegree Project: Follow Me

Me" project. Since I have only changed the image sizes and hyperparameters to adopt to the CARLA data set, the numbers and types of all layers are the same.

It is also available in github:

https://github.com/ongchinkiat/robond-follow-me

The Jupyter notebook and a HTML snapshot are in the files model_training.ipynb and model_training.html.

3 Network Architecture

A fully convolutional network (FCN) is used to train the semantic segmentation model.

The FCN contains 3 encoder blocks, followed by a 1x1 convolution layer, and 3 sysmetrical decoder blocks.

```
def fcn_model(inputs, num_classes):

# Encoder Blocks.
# Remember that with each encoder layer, the
    depth of your model (the number of filters
    ) increases.
enc1 = encoder_block(inputs, 64, 2)
enc2 = encoder_block(enc1, 128, 2)
enc3 = encoder_block(enc2, 256, 2)
# 1x1 Convolution layer using conv2d_batchnorm
    ().
onexone = conv2d_batchnorm(enc3, 256,
    kernel_size=1, strides=1)
# Decoder Blocks
dec3 = decoder_block(onexone, enc2, 256)
dec2 = decoder_block(dec3, enc1, 128)
dec1 = decoder_block(dec2, inputs, 64)
return layers.Conv2D(num_classes, 3,
    activation='softmax', padding='same')(dec1
    )
```

Each encoder block contains 1 separable convolution layer.

```
def encoder_block(input_layer, filters,
    strides):

# TODO Create a separable convolution layer
    using the separable_conv2d_batchnorm()
    function.
output_layer = separable_conv2d_batchnorm(
    input_layer, filters, strides)
return output_layer
```

Each decoder block has 1 bilinear upsampling layer, a layer concatenation step, and 1 separable convolution layers. The separable convolution layers helps to extract more spatial information from the previous layers.

```
def decoder_block(small_ip_layer,
    large_ip_layer, filters):

# TODO Upsample the small input layer using
    the bilinear_upsample() function.
upsample_layer = bilinear_upsample(
    small_ip_layer)
# TODO Concatenate the upsampled and large
    input layers using layers.concatenate
cat_layer = layers.concatenate([upsample_layer
    , large_ip_layer])
# TODO Add some number of separable
    convolution layers
output_layer = separable_conv2d_batchnorm(
    cat_layer, filters, 1)
return output_layer
```

Each decoder block has 1 bilinear upsampling layer, a layer concatenation step, and 1 separable convolution layers. The separable convolution layers helps to extract more spatial information from the previous layers.

```
def decoder_block(small_ip_layer,
    large_ip_layer, filters):

# TODO Upsample the small input layer using
    the bilinear_upsample() function.
upsample_layer = bilinear_upsample(
    small_ip_layer)
# TODO Concatenate the upsampled and large
    input layers using layers.concatenate
cat_layer = layers.concatenate([upsample_layer
    , large_ip_layer])
# TODO Add some number of separable
    convolution layers
output_layer = separable_conv2d_batchnorm(
    cat_layer, filters, 1)
return output_layer
```

The decoder includes a concatenation step which concatenates the current input with the output of a layer a few steps before it. This concatenation helps the network to retain more spatial information that were "lost" in the in between layers

Since the decoder contains a concatenation step, the size of the output of the encoders and decoders need to be the same so that there are equal number of parameters to concatenate. That is why the strides of the encoders are all

set to 2, and the bilinear upsampling in the decoder is set to 2 also.

The number of filters should increase with each layer of the encoder and decrease with each layer of decoder. The result seems to get better when I increase the number of filters. But I could not increase it beyond 64 due to the limitation of the 8GB Ram of the Nvidia GTX 1070 GPU that I am using.

4 Hyperparameters

The following Hyperparameters were used for the final run to train the FCN.

```
learning_rate = 0.01
batch_size = 8
num_epochs = 50
steps_per_epoch = 200
validation_steps = 50
workers = 2
```

I started with learning_rate = 0.01, and didn't change it because the network seems to be training well with the error reducing constantly.

I have to keep the batch_size at 8 due to the limitation of the 8GB Ram of the Nvidia GTX 1070 GPU that I am using.

The value of steps_per_epoch and validation_steps depends on the number of training and validation images available.

During training, the GPU utilization was at 100%, which is good. So I kept the number of workers at 2.

5 Quick Initial Test Run

To do a quick proof of concept, I did the absolute minimum to adopt the codes from the "Follow Me" project to this challenge.

The input RGB image size for the "Follow Me" project is 160x160. So I add in a pre-processing step to resize the CARLA images from 800x600 to 160x160 before feeding them into the network.

An initial training run of 30 epochs was done. The training result was encouraging. At least, the training loss drops consistently.

The trained model files were transferred to the Challenge Workspace (Figure 5). The cardetect.py script is used to load the model and process the input video. It was based on the demo.py script given by Udacity for this challenge. Another script cardetect-minus.py has an additional step of removing the Road pixels if a pixel is turned on for both Road and Car. This script typically gives a marginal 0.1 point improvement in the final score.

The first try of my first FCN model yield an average F score of 0.588. This got me into 4th place on the Leaderboard (Figure 6).

6 Pre-processing Improvements

The next step is to improve on the image pre-processing. The original FCN works on 160x160 images. The network receive 160x160 images as input, and output 160x160 segmentation results. Since the grading is done by comparing pixels of the final output in 800x600 pixels, getting the network to output bigger size images should yield better results.

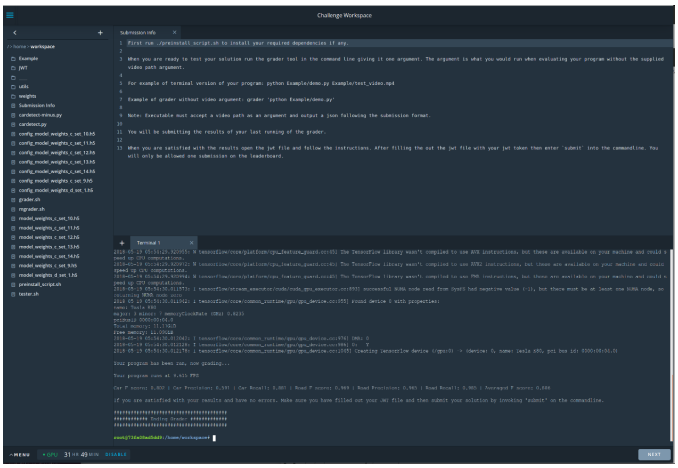


Fig. 5. Challenge Workspace

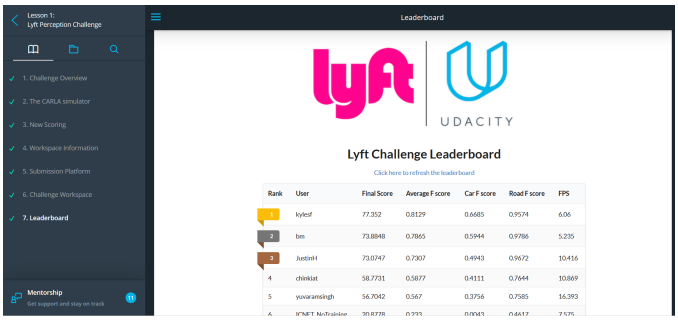


Fig. 6. Leaderboard on 5 May 2018

To avoid changes to the network architecture, I need to keep the input size to a square. First, I tried using 600x600 input images. Tensorflow complains that it runs out of memory. The 7.0GB usable Ram of the Nvidia GTX 1070 GPU is not enough. After some experimentation, I found that 280x280 works ok.

Another obvious way of improvement is to crop out the non-essential parts of the image, so that we don't waste neurons learning these non-essential parts.

For the bottom portion of the image containing the car hood, I cropped out all the pixels below row 496 initially. After looking at the result, I found that I am missing out some easy points for the road at the sides just outside of the car hood from row 496 to row 525. So I adjusted to cropped out all the pixels below row 525, and kept a small portion of the car hood. This is estimated to account for an easy 0.02 points improvement to the Road F score.

We should also crop out the top part of the image which will never contain any car or road. From a rough estimate using the tallest looking car from the given data set, row 230 seems to be the first row to have a possibility of a car.

For the width, we can observe from the given data set that car and road frequently appear at the far left and right of the image. Thus I decide to keep the whole 800 pixels for the width.

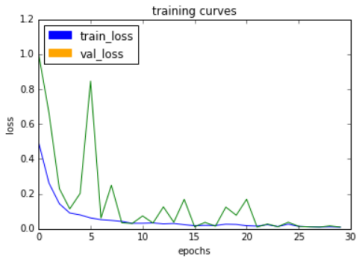
The result from this cropping is a 800x295 image. To reduce resizing errors, I decided to further crop the image to 800x280, forgoing 15 rows at the top. This makes the cropped image at 800x280, which is resized only horizon-

tally to 280x280, before feeding to the network.

After resizing to 280x280, the only other pre-processing step done is to normalize the pixel values from (0 to 255) to (-1 to 1).

Optional: changing the color space of the image from RGB to HSV / YUV, and/or adjusting the contrast curves of the image may (or may not) improve the performance of the network. I did not investigate on these due to the lack of time.

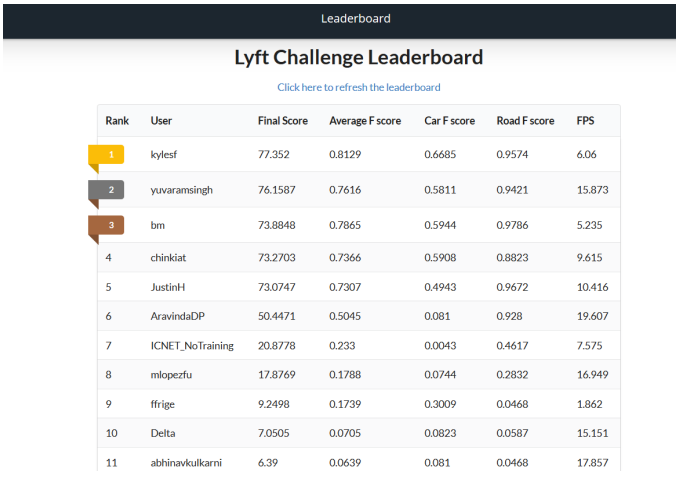
With all these pre-processing steps in place, a 30 epoch training was done. Training loss drops from 0.4979 to 0.0093 after 30 epochs, Validation loss drops from 1.0049 to 0.0100 (Figure 7)




```
200/200 [==============================] - 152s - loss: 0.0093 - val_loss: 0.0100
```

Fig. 7. Training Curve For 30 Epochs

The trained model gives a Averaged F score of 0.737, and it helped me to maintain my 4th placing in the Leaderboard (Figure 8).



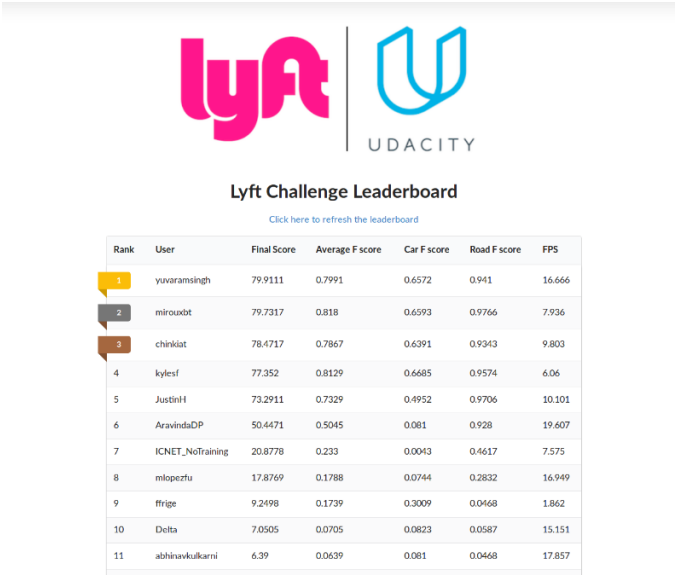
Rank	User	Final Score	Average F score	Car F score	Road F score	FPS
1	kylesf	77.352	0.8129	0.6685	0.9574	6.06
2	yuvaramsingh	76.1587	0.7616	0.5811	0.9421	15.873
3	bm	73.8848	0.7865	0.5944	0.9786	5.235
4	chinkiat	73.2703	0.7366	0.5908	0.8823	9.615
5	JustinH	73.0747	0.7307	0.4943	0.9672	10.416
6	AravindaDP	50.4471	0.5045	0.081	0.928	19.607
7	ICNET_NoTraining	20.8778	0.233	0.0043	0.4617	7.575
8	mlopezfu	17.8769	0.1788	0.0744	0.2832	16.949
9	ffrige	9.2498	0.1739	0.3009	0.0468	1.862
10	Delta	7.0505	0.0705	0.0823	0.0587	15.151
11	abhinavkulkarni	6.39	0.0639	0.081	0.0468	17.857

Fig. 8. Leaderboard on 5 May 2018 Night

7 Deep Training

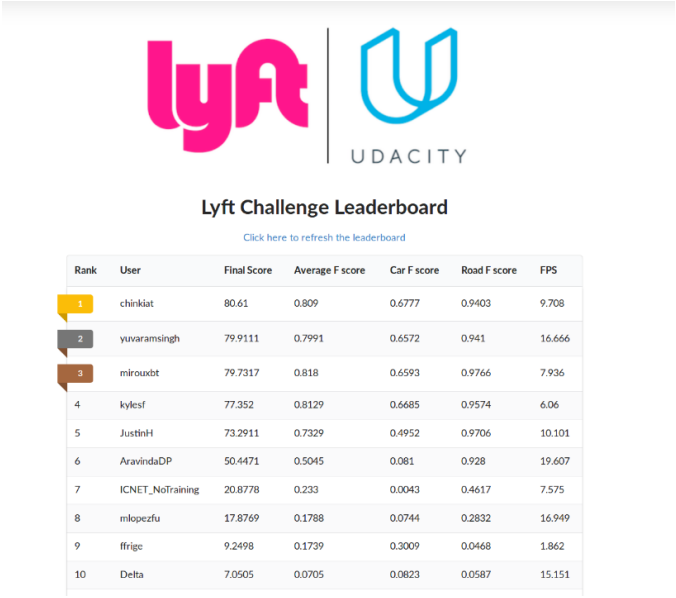
With this initial success, I did a 100 epoch training run which took around 6 hours. This gives a Final Score of 78.47. Making me the 3rd person to beat Kyle Stewart-Frantz's (kylesf) starter score of 77.35, and rank 3rd on the Leaderboard (Figure 9)

Loading back this model, and train for another few hours, I got lucky to hit a Final Score of 80.61, becoming the first person to score above 80, and got 1st place in Leaderboard ranking for the first time (Figure 10).



Rank	User	Final Score	Average F score	Car F score	Road F score	FPS
1	yuvaramsingh	79.9111	0.7991	0.6572	0.941	16.666
2	mirouxbt	79.7317	0.818	0.6593	0.9766	7.936
3	chinkiat	78.4717	0.7867	0.6391	0.9343	9.803
4	kylesf	77.352	0.8129	0.6685	0.9574	6.06
5	JustinH	73.2911	0.7329	0.4952	0.9706	10.101
6	AravindaDP	50.4471	0.5045	0.081	0.928	19.607
7	ICNET_NoTraining	20.8778	0.233	0.0043	0.4617	7.575
8	mlopezfu	17.8769	0.1788	0.0744	0.2832	16.949
9	ffrige	9.2498	0.1739	0.3009	0.0468	1.862
10	Delta	7.0505	0.0705	0.0823	0.0587	15.151
11	abhinavkulkarni	6.39	0.0639	0.081	0.0468	17.857

Fig. 9. Leaderboard on 6 May 2018



Rank	User	Final Score	Average F score	Car F score	Road F score	FPS
1	chinkiat	80.61	0.809	0.6777	0.9403	9.708
2	yuvaramsingh	79.9111	0.7991	0.6572	0.941	16.666
3	mirouxbt	79.7317	0.818	0.6593	0.9766	7.936
4	kylesf	77.352	0.8129	0.6685	0.9574	6.06
5	JustinH	73.2911	0.7329	0.4952	0.9706	10.101
6	AravindaDP	50.4471	0.5045	0.081	0.928	19.607
7	ICNET_NoTraining	20.8778	0.233	0.0043	0.4617	7.575
8	mlopezfu	17.8769	0.1788	0.0744	0.2832	16.949
9	ffrige	9.2498	0.1739	0.3009	0.0468	1.862
10	Delta	7.0505	0.0705	0.0823	0.0587	15.151

Fig. 10. Leaderboard on 6 May 2018 Night

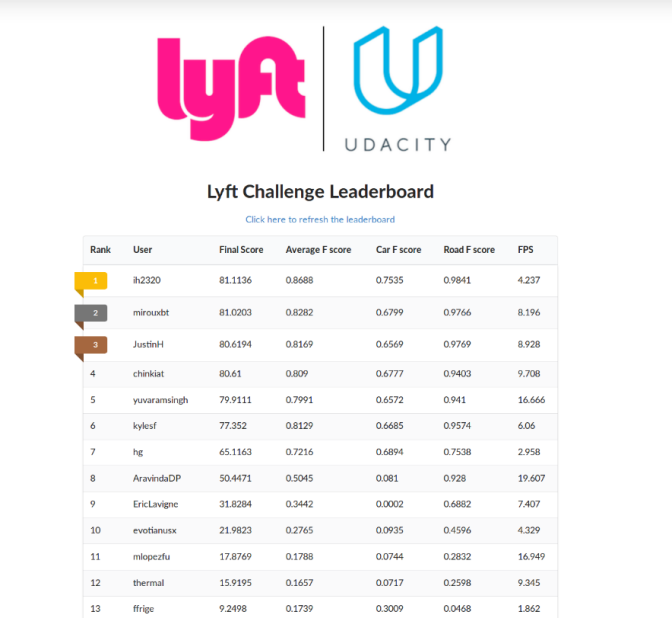
I stayed at first place for slightly more than 1 day, and was overtook by 3 other contestants on 8 May 2018. (Figure 11)

8 Toying Around

From 7 May to 11 May (Monday to Friday), I was busy with my full time job. So I only spent some time toying around with data augmentation and hyperparameter tuning.

Some data augmentation techniques I tried includes generating another set of the given images with slightly brighter or darker colors (+ or - some values for each pixel), change the color of the cars in the image (for those pixels labeled car in the ground truth, change the RGB value to red/blue/green/etc).

Sometimes they give better results, sometimes worst. But because I was doing continuous training of the model



Lyft Challenge Leaderboard

Click here to refresh the leaderboard

Rank	User	Final Score	Average F score	Car F score	Road F score	FPS
1	ih2320	81.1136	0.8688	0.7535	0.9841	4.237
2	mirouxbt	81.0203	0.8282	0.6799	0.9766	8.196
3	JustinH	80.6194	0.8169	0.6569	0.9769	8.928
4	chinkiat	80.61	0.809	0.6777	0.9403	9.708
5	yuvaramsingh	79.9111	0.7991	0.6572	0.941	16.666
6	kylesf	77.352	0.8129	0.6685	0.9574	6.06
7	hg	65.1163	0.7216	0.6894	0.7538	2.958
8	AravindaDP	50.4471	0.5045	0.081	0.928	19.607
9	EricLavigne	31.8284	0.3442	0.0002	0.6882	7.407
10	evotianusx	21.9823	0.2765	0.0935	0.4596	4.329
11	mlopezfu	17.8769	0.1788	0.0744	0.2832	16.949
12	thermal	15.9195	0.1657	0.0717	0.2598	9.345
13	ffrige	9.2498	0.1739	0.3009	0.0468	1.862

Fig. 11. Leaderboard on 8 May 2018

by loading the previous trained model before starting the training, I cannot be sure if the improvements were due to more epochs of training, or due to the data augmentation.

Anyway the improvements were marginal, and the best score I got for a whole week of toying around was 81.7, just 1.1 points higher than just using the given data set.

9 CARLA Simulator

Since the start of this challenge, I was also busy working on the last 2 projects for the Robotics Nanodegree Term 2. I managed to submit my Robotics Project 4 on 4th of May, and the last project Robotics Project 5 on 12th of May, to complete the Robotics Nanodegree on 13th May.

After I submitted the last project on 12th May, it is time to download the CARLA Simulator.

The simulator program runs ok on my desktop. The example script client_example.py looks simple. I placed the 2 camera settings according to the parameters provided by kylesf.

```
camera0 = Camera('CameraRGB')
camera0.set_image_size(800, 600)
camera0.set_position(1.30, 0, 1.30)
settings.add_sensor(camera0)

camera1 = Camera('CameraSeg', PostProcessing='
    SemanticSegmentation')
camera1.set_image_size(800, 600)
camera1.set_position(1.30, 0, 1.30)
settings.add_sensor(camera1)
```

Then it is just simply reading the data for each frame and save them as PNG files.

```
imageRGB = sensor_data['CameraRGB'].data
imageSeg = sensor_data['CameraSeg'].data
```

With some adjustments to the parameters, the script is ready for batch generation of data. I observed that for the first 30+ frames, the car was apparently dropping down to the road. So I always skipped the first 40 frame before saving the frames to disk.

I generated a few batches of data, and started to train my network using the new data. While waiting, I packaged 1000 frames (10 runs of 100 frames) into a ZIP file, and put on Github, becoming the first (and only?) competitor to release data for public use.

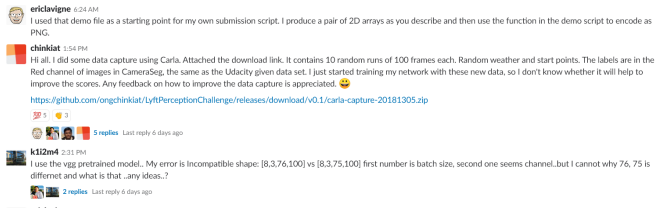


Fig. 12. Data Set Annoucement on Slack Channel

The act of releasing data to help other fellow competitors paid off immediately. Eric Lavigne looked at it and suggested that I turn up the number of vehicles value (Figure 13). As a feedback to his helpful tip, I generated and released another set of 1000 frames data, with number of vehicles bumped up to 80.

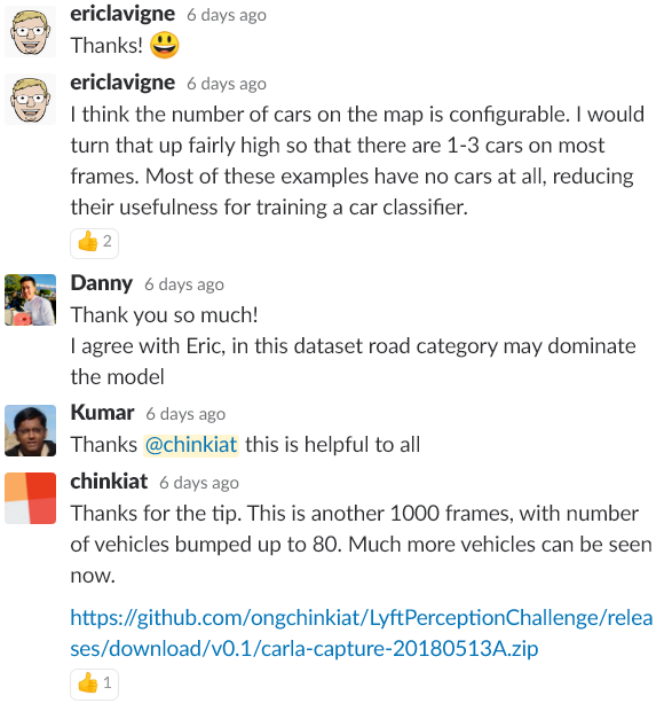
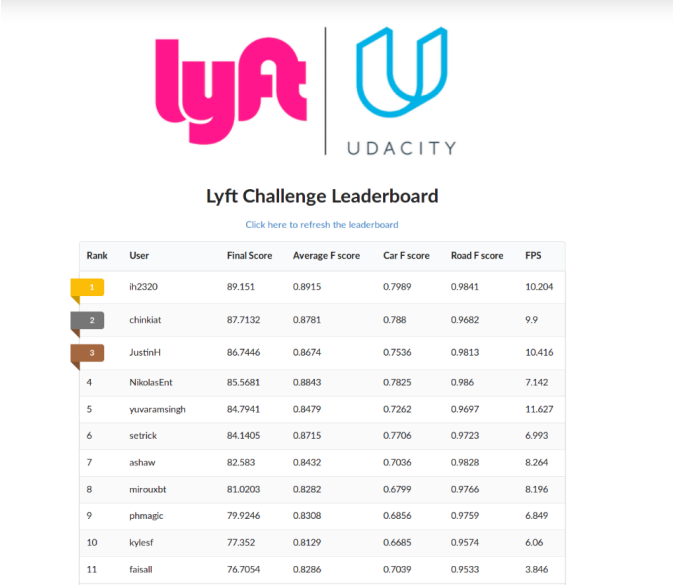


Fig. 13. Configuration Suggestion From Eric Lavigne

I generated more training data with high number of vehicle values and added them to my training set.

10 Results With More Data

By the time I got the model with more CARLA data trained, my ranking has dropped off the top 5. In less than 1 day of training with fresh captured data from CARLA, the score jumped from the 81-83 range to 87.71, putting me back to 2nd place (Figure 14).



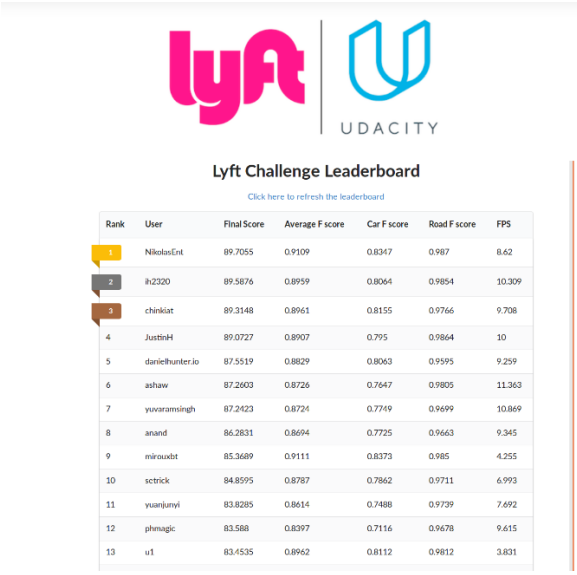
Lyft Challenge Leaderboard

Click here to refresh the leaderboard

Rank	User	Final Score	Average F score	Car F score	Road F score	FPS
1	ih2320	89.151	0.8915	0.7989	0.9841	10.204
2	chinkiat	87.7132	0.8781	0.788	0.9682	9.9
3	JustinH	86.7446	0.8674	0.7536	0.9813	10.416
4	NikolasEnt	85.5681	0.8843	0.7825	0.986	7.142
5	yuvaramsingh	84.7941	0.8479	0.7262	0.9697	11.627
6	setrick	84.1405	0.8715	0.7706	0.9723	6.993
7	ashaw	82.583	0.8432	0.7036	0.9828	8.264
8	mirouxbt	81.0203	0.8282	0.6799	0.9766	8.196
9	phmagic	79.9246	0.8308	0.6856	0.9759	6.849
10	kylesf	77.352	0.8129	0.6685	0.9574	6.06
11	faisall	76.7054	0.8286	0.7039	0.9533	3.846

Fig. 14. Leaderboard on 13 May 2018

It is weekdays again, and another few days of toying around with the data sets and burning GPU cycles for (hopefully) better scores. Improvements were again marginal. Now (19 May), my best score is currently 89.31, ranking 3rd (Figure 15).



Lyft Challenge Leaderboard

Click here to refresh the leaderboard

Rank	User	Final Score	Average F score	Car F score	Road F score	FPS
1	NikolasEnt	89.7055	0.9109	0.8347	0.987	8.62
2	ih2320	89.5876	0.8959	0.8064	0.9854	10.309
3	chinkiat	89.3148	0.8961	0.8155	0.9766	9.708
4	JustinH	89.0727	0.8907	0.795	0.9864	10
5	danielhunter.io	87.5519	0.8829	0.8063	0.9595	9.259
6	ashaw	87.2603	0.8726	0.7647	0.9805	11.363
7	yuvaramsingh	87.2423	0.8724	0.7749	0.9699	10.869
8	anand	86.2831	0.8694	0.7725	0.9663	9.345
9	mirouxbt	85.3689	0.9111	0.8373	0.985	4.255
10	sctrick	84.8595	0.8787	0.7862	0.9711	6.993
11	yuanjunyi	83.8285	0.8614	0.7488	0.9739	7.692
12	phmagic	83.588	0.8397	0.7116	0.9678	9.615
13	u1	83.4535	0.8962	0.8112	0.9812	3.831

Fig. 15. Leaderboard on 17 May 2018

11 VISUALIZATION

To have a better understanding of the performance of my FCN, I ran it through some images that were not used in training. The result is made into a movie and posted on Youtube:

https://www.youtube.com/watch?v=hMZ135eSIS8

It is also featured in the Sharing Zone (Figure 16).

In Visualization Screen 1 (Figure 17), we see that the network can identify the car and road generally, but there were quite a bit of false positives around the car area.

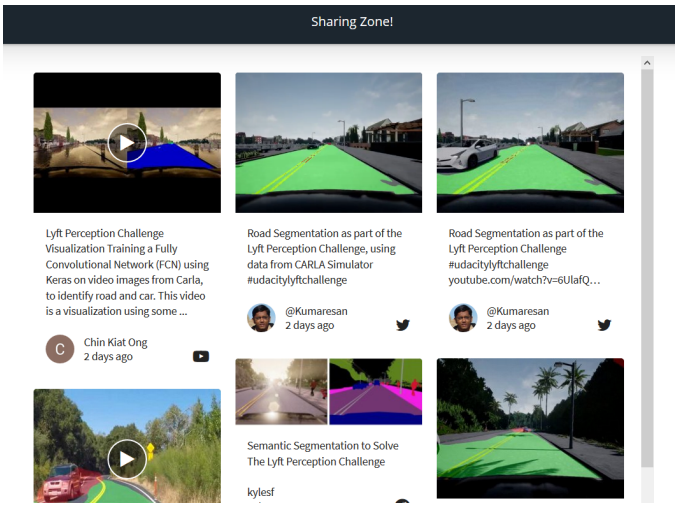


Fig. 16. Posting On Sharing Zone

Fig. 17. Visualization Screen 1

In Visualization Screen 2 (Figure 18), we see that the car is only partially identified. Half the car is not marked green. This should mean that my training set don't have enough frames of this particular grey color car, and I can probably improve the car detection score by feeding the network with more data.

One observation after playing around with the CARLA simulator for a few days is that there are probably less than 100 different car models in the simulator. The same "red Coca Cola truck", "blue with white top mini bus", "black and white police car", etc is seen again and again. I suspect that given a huge enough training data from the CARLA simulator, and a huge enough neural net, it is possible to "memorize" all the cars available in the simulator.

Throughout the video, we can see patches of green and blue at areas outside of the "normal" zone. Example, the right side of Figure 19. These are fairly large patches, and they are pulling down the precision scores.

We can probably do some post processing to identify these patches, using some standard computer vision technique, and drop off those patches which is deemed to be low chance to be road or car. Currently, no work is done on this, due to the lack of time.

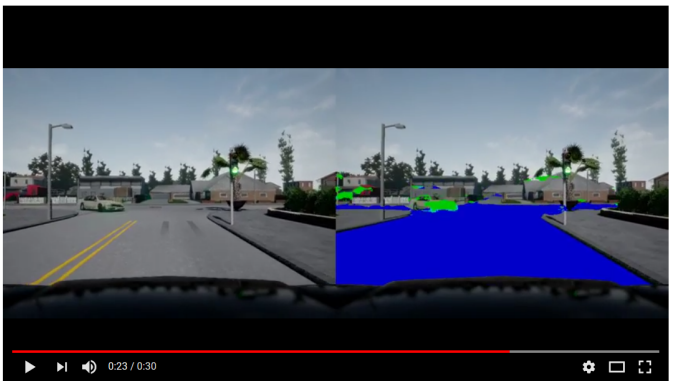


Fig. 18. Visualization Screen 2

Fig. 19. Visualization Screen 3

12 Inference Optimization

I read about a tip from Slack #lyft-challenge channel. Faisal Khan (faisall) commented that we can pass multiple frames to the model during inference (Figure 20).

I changed my code to batch process the incoming frames, and tried the grader. The FPS increased from around 9.8 to around 11.2. That is an increase of up to 0.2 in the final score, and I have a bit of buffer for additional processing.

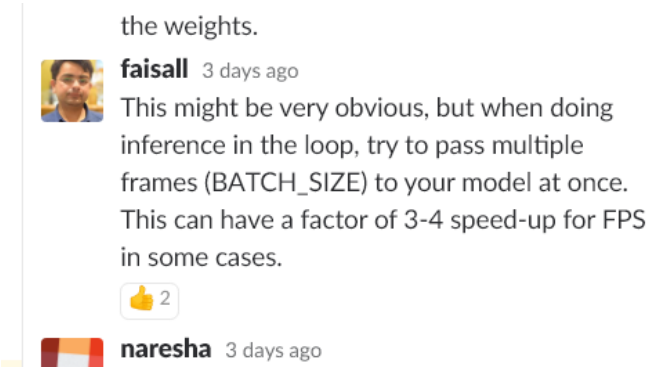


Fig. 20. Faisal Khan Suggesting Batch Processing During Inference

13 Post Processing Threshold Values

A few days pass by, where I am just feeding the network with thousands and more thousands of images, and only getting marginal improvements. I revisited the inference script, and thought that I should play around with the threshold values for the output of the network.

My original code looks like this:

```
# get the car pixels from the Green channel
green = pred[:,:,1]
# pixels > 95% is car
green_bit = np.where(green>0.95,1,0).astype('
    uint8')

# get the road pixels from the Blue channel
blue = pred[:,:,2]
# pixels > 95% is road
blue_bit = np.where(blue>0.95,1,0).astype('
    uint8')
```

I had the misconception that the prediction result values range from -1 to 1.

I changed the threshold values to 0.5, and got the same scores. Then I realised something is wrong. A check on the values confirmed that they range from 0 to 255. I changed the threshold value to 250, and got some improvements in the road score, but lower score for cars. Thus I decide to give different threshold values for road and car.

After playing around with some numbers, I found the highest score when car threshold = 0 and road threshold = 250. This simple change improves my overall score by 0.8, and made me the first to cross 90 points in the Leaderboard (Figure 21).

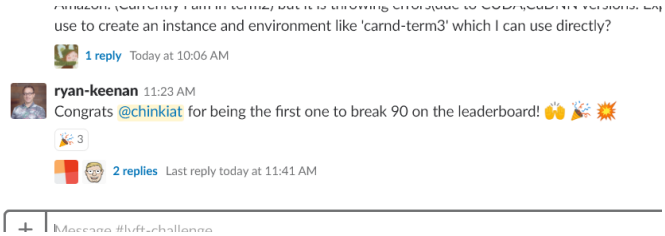


Fig. 21. First to cross 90 points

Some days later, I revisited this topic, and made some recordings of the result in Table 1. As we increase the car threshold value from 1 to 250, car recall drops from 88% to 66.6%, while precision increase from 69% to 89.5%.

Checking with the scoring formula for this challenge, $\beta = 2$ for vehicles, favouring recall more than precision. Thus my choice of car threshlod = 1 is the result of the scoring formula. My choice of road threshold high at 250 is also consistent with the $\beta = 0.5$ for road, favouring precision more than recall for road.

The interesting observation on this is that the Average score drop from 91.0 (which will be ranked top 10 on 29 May) to 84.4 (rank 38 on 29 May), using the same network with just a different choice of car threshold value. It may be that everyone with score above 75 have similar network performance, just slightly different choice of parameters.

14 Keep Feeding The Network

Since I started using CARLA Simulator to get more training images on 13 May, I had been repeatedly generating a few thousand new images and feed them to the network.

Car Threshold	Car Recall	Car Precision	Car F Score	Average F Score
1	0.880	0.690	0.834	0.910
50	0.822	0.813	0.820	0.903
100	0.799	0.838	0.807	0.896
150	0.776	0.856	0.791	0.888
200	0.746	0.871	0.768	0.877
250	0.666	0.895	0.702	0.844

TABLE 1
Effect of Threshold on Scores

Sometimes, I generate a batch of images with more cars. Sometimes, with less cars. Sometimes, with more pedestrians.

Results from the additional images had been marginal. Sometimes, it gets better scores, sometimes lower. But overall, it tends to get better.

Till 29 May, the network have ”seen” more than 40 thousand images.

This effort seems to have paid off with the announcement on 27 May that Udacity ”will be running everyones code against a new and never-before-seen dataset”. Hopefully, my network will have seen enough of the CARLA environment to score well on this new dataset.

15 Discussion And Conclusion

It is fun participating in this Lyft Perception Challenge.

I am lucky to own a PC with GTX 1070 GPU, which I bought specifically to work on Udacity Deep Learning Foundation, Robotics and Self Driving Car Nanodegree projects. This allowed me to continuously train the FCN without worrying about per hour rental costs for Cloud GPU services.

I learned a lot in the process. Just by reading this report, it may looked like every next step I made turned out well. In reality, there are a lot of missteps and silly mistakes made along the way.

The biggest gamble I made was in the choice of network architecture. It was very tempting to start with a well known and proven network like the VGGNet. In comparison, the network I used was an Udacity project assignment I coded myself base on skeleton code provided.

The starting skeleton code looked like this:

```
def encoder_block(input_layer, filters,
    strides):

      # TODO Create a separable convolution
          layer using the
      # separable_conv2d_batchnorm() function.

      return output_layer

def decoder_block(small_ip_layer,
    large_ip_layer, filters):

      # TODO Upsample the small input layer
          using the bilinear_upsample()
          function.

      # TODO Concatenate the upsampled and
          large input layers using
      # layers.concatenate.

      # TODO Add some number of separable
          convolution layers.

      return output_layer

def fcn_model(inputs, num_classes):

      # TODO Add Encoder Blocks.
      # Remember that with each encoder layer,
           the depth of your model (the number
           of filters) increases.

      # TODO Add 1x1 Convolution layer using
          conv2d_batchnorm().

      # TODO: Add the same number of Decoder
          Blocks as the number of Encoder
          Blocks


      # The function returns the output layer
          of your model. "x" is the final
          layer obtained from the last
          decoder_block()
      return layers.Conv2D(num_classes, 1,
          activation='softmax', padding='same
          ')(x)
```

There was a lot of room for free play. I was free to choose the number of layers, the depth and width of each layer, the type of operations between each layer, etc. There is bound to be some places where I may have chosen a non-optimal value and may thus limit the performance of the whole network.

I followed my gut feel and go with my own network, hoping that my few weeks experience working with this network in my previous project would enable me to adopt well with the problem on hand.