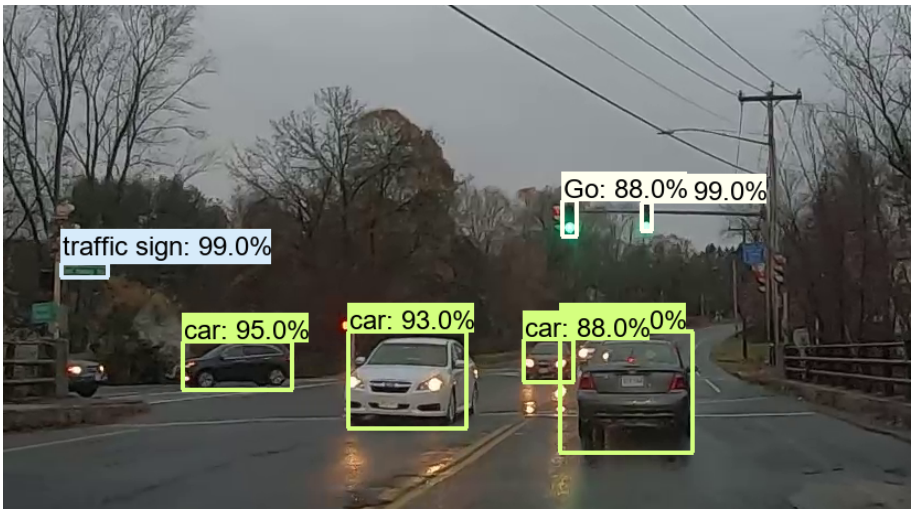
Traffic Vision

The First Step To Autonomous Driving

Wenbo Li, Baitian Zhang

November 2020



Contents

1 Introduction

1. What Is Traffic Vision
   Traffic Vision is a traffic object detector with the use of Tensorflow Object Detection API[1] based on TensorFlow 2[2]. It detects the objects on roads such as cars, trucks, pedestrians, motorcycles, by constantly analyzing the image. Our Traffic Vision can detect 12 different classes of objects. Input source can be static images or videos. It can also read camera feed to give real-time detections. With a NVIDIA GPU that has CUDA compute capability[3] greater or equal to 7.5, Traffic Vision can perform at about 15fps. See this link for CUDA capability details: CUDA Compute Capability List[3]

2. Inspirations
   In the 21st century, new technologies are invented everyday, through all aspects of our daily life. For many of these technologies, object detection is the first step. We train computers to see and recognize the world like humans. With this capability, we can then train computers act like humans. In this case, we are looking at autonomous driving. Autonomous driving based on LiDAR has been developing for decades[4], for example, the Google Waymo[5], but transformation from technology to actual products takes years. Only in recent years or even months, we started to see the autonomous driving technologies based on computer vision being deployed to production cars, in this case, Super Cruise from General Motors[9], NIO Pilot from NIO[6], XPILOT from XiaoPeng[7] and Autopilot from Tesla[8].

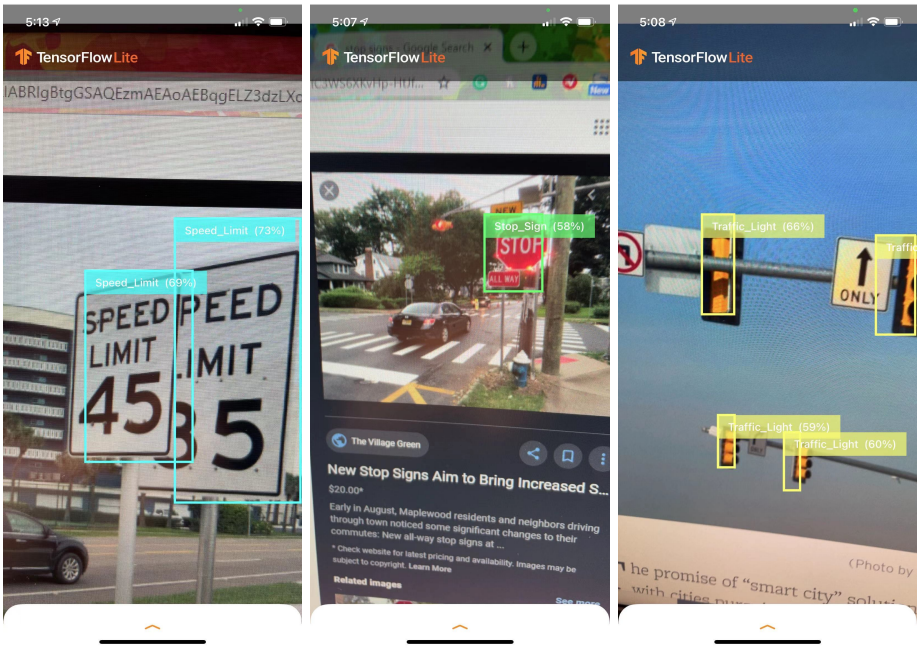
Figure 1: How Tesla understands an intersection[11]

You might think why. Why Waymo cars can only be running in Phoenix, Arizona but not in the other places of the world even with decades of development? That's a competition about LiDAR versus ordinary cameras, and I will not discuss that huge topic here. With the great passion for technologies and cars, we decide to try this very first step for autonomous driving: detection on traffic objects.

2 Before The Actual Project

We planed to train our model on PC and deploy it to iPhone to perform real-time detection. Before the beginning of the actual project, we thought it would be better to play around with TensorFlow[2] and the API[1]. Our goal was to set up the environment and went through all steps successfully so we would be familiar with the whole thing and do better in the actual project.

We started with only 3 target classes: traffic lights, stop signs and speed limits. We downloaded 500 images for each class from Google Image[12] and put 90% for train, 10% for test. We then labeled them and converted them to tfrecord. We chose SSD MobileNet since it's fast. After 1000 Epochs of training, we put our model in the TensorFlow Lite template app[13]. We'll talk about how we did the labeling, converting and training, deploying in the later sections of the report. Below are the results.



After this, we broke though all the steps and we were ready to start project Traffic Vision.

3 Preparing Dataset

For academic purpose, we don't need a huge amount of industrial-level data. So what we did first was collecting some lightweight dataset. Then we found the performance of our trained model was really poor. We realized that this project requires a huge amount of data to reach a semi-prefect performance even with

academic purposes only. We have thought of several ways of preparing it. Here are what we did.

3.1 Choose/Create Dataset

An ideal training dataset should include target objects at/with different angles, different lighting, image quality, distances (object sizes), etc.

3.1.1 Create New Data

The initial thought was to record our own video, manually label the video frame by frame. Therefore we sat up two GoPro cameras in a car, drove in Amherst during day and night to collect data.

We git cloned labelImg from GitHub[14] to label images. Labeled data will be saved as XML files.

It turns out our video quality was good as expected, but manually picking out thousands of frames and labeling them was time-consuming. It is unachievable, given we only have a limited amount of time. We decided to give up on this but we saved the video for future testing purpose.

3.1.2 Download From Google Image[12]

Instead of doing a huge amount of labeling work, we went for Google Images[12] like we what did in the pre-project. We downloaded images of traffic signs, cars, pedestrian, etc.. We found two problems with this approach: 1. All the pictures needs to be downloaded manually (even with many chrome extension plugins), and the irrelevant images are meant to be removed manually as well, which brings heavy burden, reduces our speed. 2. A lot images are overly standard, which do not represent the real situation or what people see in a moving car. Insisting on this approach, we expect a bad performance of the trained model.

3.1.3 Search For Open Dataset

Trying the previous two approaches and not getting ideal results, we began looking for the open dataset provided by companies and institutions. We found several organizations and institutions doing this work:

1. Astyx Dataset HiRes2019[15]
2. Level 5[16]
3. nuScenes Dataset[17]
4. Open Images V5[18]
5. Oxford Radar RobotCar Dataset[19]
6. Pandaset[20]
7. Waymo Open Dataset[21]
   We did not choose the above datasets since they either are made for LiDAR or with 3D object detection labels which does not meet our requirements.
8. Berkeley Deep Drive[22]
   We finally choose BDD100k for training and testing since it contains everything we want.

[22]

3.2 XML/json to CSV

1. XML to CSV
   When we use labelImg from GitHub[14] to label images, we got XML files. XML files contain coordinates of objects in the image and the class names of them. We use *xml_to_csv.py* to convert XML to CSV. Please see *xml_to_csv.py* for details.
2. json to CSV
   After we downloaded BDD100k[22] dataset, we noticed the labels are saved in json files, which have nested json format and were a little bit complicated. So we inspected the json carefully and filtered out parts we need. Then we flattened the nested json with information we need and discarded data that we don't need. Again, we translated all the flattened json into a CSV file. Please see *json2csv.py* for details.

3.3 Define Classes

We inspected and filtered the pre-labeled annotations and sat 12 classes for our project:

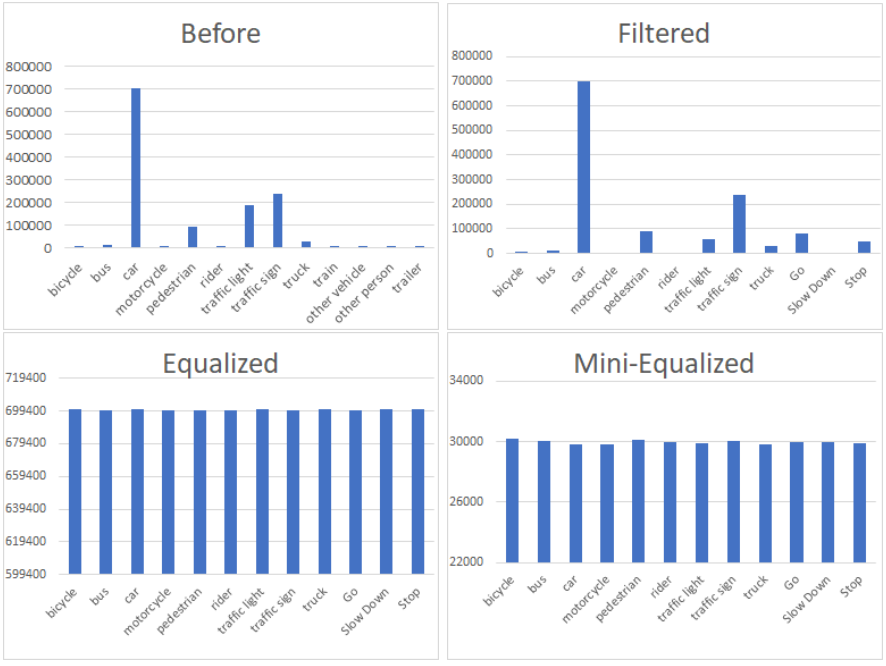
Where *Go* means green traffic lights. *Slow Down* means yellow traffic lights.

id	Label	id	Label	id	Label	id	Label
1	bicycle	4	Go	7	rider	10	traffic light
2	bus	5	motorcycle	8	Slow Down	11	traffic sign
3	car	6	pedestrian	9	Stop	12	truck

Stop means red traffic lights. And *traffic light* means unknown color or side or back of traffic lights.

3.4 Equalizing Data Size

After translated json to CSV, we found sample size of classes are highly uneven which can lead to poor detection confidence of classes that has less training data, especially in CNN. So we super sampled classes with less data, making all classes as equal as possible. And the training data size was tremendously enlarged. Considering Baitian doesn't have a PC with CUDA GPU and Wenbo's PC has only one RTX 2080Ti, we decided for a small balance size to make the training process faster. See below for details.



3.5 CSV to Tfrecord

Tensorflow Object Detection API[1] takes in .tfrecord files as training and testing data, so we need to convert CSV file to tfrecord. We use *generate_tfrecord.py* to convert XML to CSV. Please see *generate_tfrecord.py* for details.

4 Select Model

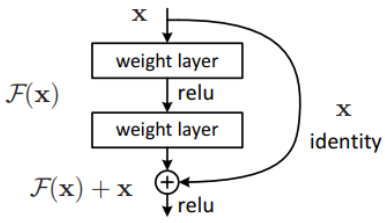
TensorFlow 2 Detection Model Zoo[23] provides people with pre-trained models. With so many available selections, we should aim for both speed and performance. If we choose a model that is relatively slow to process a frame, we won't get a real-time like effect. If we choose a model that has unstable and poor detection scores (confidence), this project becomes meaningless.

4.1 SSD MobileNet

MobileNet has great advantages in speed.
The traditional convolution network has the computing cost: DK · DK · M · N · DF · DF, DF is the spatial width and height of a square input feature map1, M is the number of input channels (input depth), DG is the spatial width and height of a square output feature map and N is the number of output channel (output depth),DK is the spatial dimension of the kernel assumed to be square and M is number of input channels[24]
Depthwise separable convolution are made up of two layers: depthwise convolutions and pointwise convolutions. Its computing cost is: DK · DK · M · DF · DF, therefore extremely efficient compared to traditional Convolutional network[24]. It is very suitable for deploying on portable platforms like Android and ios. That is why we chose MobileNet as our "before the actual project" model. It turns out fast and stable on a iPhone.

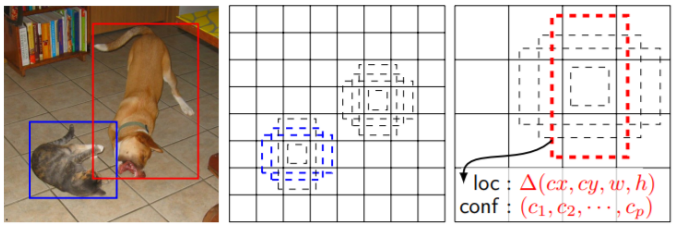
4.2 SSD ResNet50

ResNet is one kind of the CNN. Since it was published, it has won the first places in five main competitions, proving its extraordinary performance of image classification and detection.
Resnet alleviates a long-existing problem: the degradation problem.
Deeper neural networks are more difficult to train[25]When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, but the vanishing of gradients. Adding more layers to a suitably deep model leads to higher training error
ResNet, or Residual Neural Network, uses residual values to train the network[25].



As shown in the graph, the input is X, the two layers calculates. The original mapping is recast into F(x)+x using this shortcut connection.[26] This learning process learns less content, and its orignal input is transferred without loss. Compared to MobileNet, ResNet has higher overall accuracy. Since we are deploying our model on the PC, the accuracy is considered beyond computational cost.

SSD, Single Shot MultiBox Detector, is a one stage object detection method. One stage means it is sampling on only one picture, and MultiBox means it can detect multiple objects in one picture.This method requires ground truth boxes that are given by humans as in the pictures.



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

The SSD sets a small set of default boxes of different ratios at each location with several feature maps with different scales (e.g. 8×8 and 4×4 in (b) and (c)). And for each default box it predicts the shape offsets and the confidences for all categories by comparing with the ground truth boxes. Then a probability value will be calculated. The category with the highest probability value will be the prediction.[40] The SSD Resnet-50 is outputting four things: Bounding boxes of Objects, Class Probabilities, Object names, and different colors for different bounding boxes.

For the bounding box choice, the SSD can self-adjust the ratio and size of boxes. The training objective of SSD is to match the predicted box with the ground truth box as close as possible. For the ground truth box, we have a gradient (cx,cy,w,h). cx and cy are the center of the box, and w, h stands for

width and height. The predicted box is the transformation from the truth box outline to the truth box's four coordinates. These coordinates are:
[40]

$$g_j^{cx} \quad g_j^{cy} \quad g_j^{w} \quad g_j^{h}$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^{w} \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^{h}$$
$$\hat{g}_j^{w} = \log\left(\frac{g_j^{w}}{d_i^{w}}\right) \qquad \hat{g}_j^{h} = \log\left(\frac{g_j^{h}}{d_i^{h}}\right)$$

During the prediction, the true box outline can be obtained by reversing the process above:

$$g^{cx} = g_{jsub}^{cx} \cdot d_i^{w} + d_i^{cx} \qquad g^{cy} = g_{jsub}^{cy} \cdot d_i^{h} + d_i^{cy}$$
$$g_j^{w} = d_i^{w} exp\left(g_{jsub}^{w}\right) \qquad g_j^{h} = d_i^{h} exp\left(g_{jsub}^{h}\right)$$

For the categories: We set 12 different colors. The code will assign the corresponding color to their bounding boxes.

5 Train Model

5.1 Configuration

Pre-trained models from TensorFlow 2 Detection Model Zoo[23] come with a pre-configured *pipeline.config*. To train our own dataset, we need to make some modifications to it.

Our model SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)[27] fills about 5.3GB in GPU's VRAM. Since there are 11GB VRAM in a RTX 2080Ti, we can only set batch size = 2. Out-of-memory error will be throw if batch size if bigger. After several trains, we found choosing cosine decay function as learning rate with base = 0.04 gives best gradient decay graph. For loss function, we choose ReLU as activation function as it is faster and can prevent gradient vanishing, comparing to sigmoid. We have 30000 training samples for each class, with batch size = 2. To get one Epoch, we need

$$\texttt{number of steps} = \frac{30000 \times 12}{2} = 180000 \tag{1}$$

For each step, RTX 2080Ti takes about 0.35s. So we need

$$\texttt{time to run 1 epoch} = 180000 \times 0.35 = 63000s = 17.5hours \tag{2}$$

5.2 Start Training

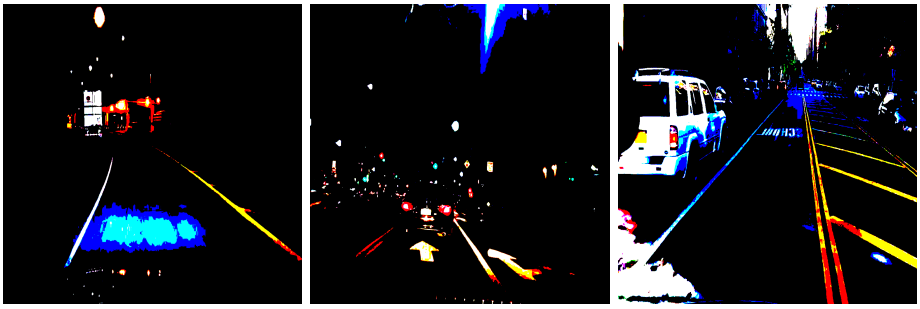
After we got tfrecords and *pipeline.config* set up, we can start training. We trained on our own PC, using the *model_main_tf2.py* which is provided by TensorFlow Object Detection API[1]. Here are terminal commands you need to give to start training.

```
$ python model_main_tf2.py
--model_dir = MODEL_OUTPUT_DIR
--pipeline_config_path = PATH_TO_PIPELINE.CONFIG
```

And use this command to watch current training status:

```
$ tensorboard --logdir = MODEL_OUTPUT_DIR
```

In tensorboard we can watch loss, steps, learning rate and more. This is how training data looks like



6 Export Model

After training is finished, we need to save the trained model or convert it to a TensorFlow Lite Model. Note that Tensorflow 2 and TensorFlow Lite have different formats for model.

6.0.1 Export As TensorFlow 2 Model

We used the *exporter_main_v2.py* provided by TensorFlow Object Detection API[1] to export trained checkpoints to a Tensorflow 2 model.

6.0.2 Export As TensorFlow Lite Model

To use this model on portable platforms, we need to convert Tensorflow 2 model to a TensorFlow Lite Model. We followed tf.lite.TFLiteConverter[29] to create a *tf2tflite.py* and used it to do the converting. See *tf2tflite.py* for details.

7 Performance Comparison

Here are samples of prediction result before and after fine tuning, using the test data.

8 Deploy Model

After We got our saved model, we can use it to do some detection tests. Firstly, we need to deploy the model to a driver program that can run the model.

8.1 On IOS platform

Using IOS image detection example[28] we are able to simply deploy the trained model to iPhone without wasting time creating an app from sketch. After created *cocoapod* environment, we *-pod update* the packages to the latest version and replaced the model file and *labelmap.pbtxt* file with our own ones. Then the app was ready to run! This is how we did in "before the actual project".

However, there are always many unexpected errors and bugs in a development process. In this case, our ResNet model gave weird outputs. It only gave detections on class id = 1 and id = 11 with scores = 0.01. The weird outputs won't change whatever we did. Then we found the GitHub issue: SSD ResNet from model zoo not working after conversion to TFLite[30], Which describes exactly same issue we were facing. After some research, we found this bug has been on for months. Instead of waiting for TensorFlow Lite team to fix it, we chose to give up on deploying to portable devices.

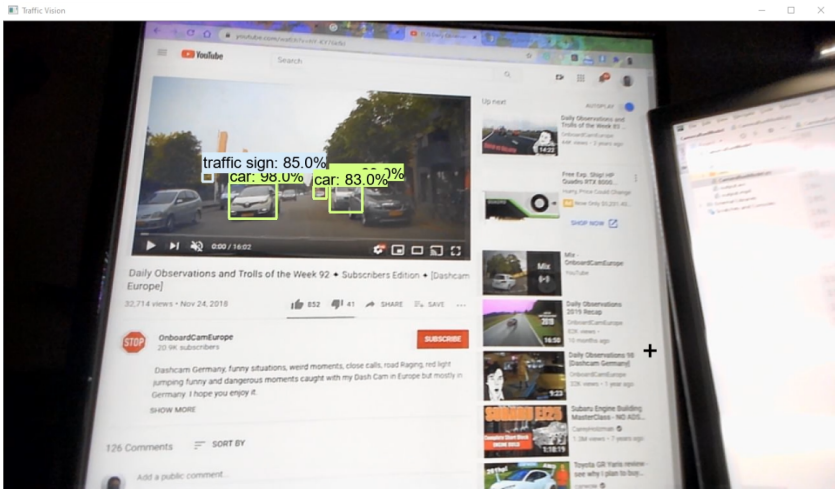
8.2 On Windows

On desktop, we have larger flexibility. Thanks to CUDA, computing can be several times faster than only using CPU or an iPhone. We import our testing images (or videos) to our model using *opencv-python*[31] package. The model

processes frames and sends back detection classes, boxes, and scores. Then we simply mark them on the frames and output to user.

8.2.1 Testing With Webcam

We used OpenCV[31] to read input frames from a webcam. We played a YouTube video[32] on another monitor and pointed the webcam to that monitor.



8.2.2 Testing with PC In A Car

We wanted to test our detector's performance in real life, just like what Tesla is doing. Therefore we move this desktop PC (with a GTX 1070Ti) and a monitor onto a car.

The 12v cigarette lighter plug was built in with a 12A fuse, 150 watt is all we can draw from that, but our desktop consumes 300+ watts at running. So the problem is having enough power. We bought a new car battery from Walmart and connected it to a 500 Watts DC-AC inverter. Theoretically, this setup is enough to power the PC for at least 3 hours. We excitedly plugged computer's power cord into the inverter. As soon as the power was turned on (only the power button, not the computer), the motherboard started buzzing. A few seconds later, the inverter turned off. We think this is a electric wave issue that the inverter may not producing pure sin waves since computer's PSU requires AC with pure waves to function normally. So it will not let it boot up with such impure electricity wave, for protection purpose. The instantaneous peak power draw at boot up stage was another factor blocking it from running. We could have solved the problem by using 18650 batteries and an inverter of larger capacity, but they were sold out in Amherst local store. Finally, we decided not to do the PC-In-Car test.

8.2.3 Testing With Live Streaming

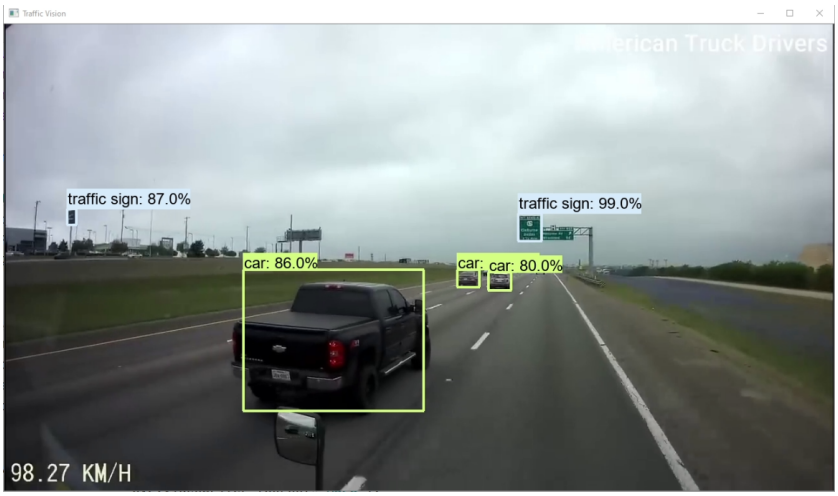
Then we tried to use a mobile live streaming app, like YouTube or Twitch, to send live frames back to the PC by putting the phone on windshield like a dashcam. We finally gave up on this since OpenCV[31] can't read live streaming frames smoothly and stably. Also, cellular coverage in Amherst is poor and the video quality is bad even with 5G connection.

8.2.4 Testing With Recorded Video

However, OpenCV can read video files smoothly and stably. Here we used the video shoot in "Create New Data". See *CameraRunModel.py* for details, and see "Example Runs" section for result.

8.2.5 Testing With YouTube Video

OpenCV can also read URLs of YouTube videos smoothly and stably. Here we used the video: Wrong-way driver hits a semi and 3 other cars[33] as test. See *CameraRunModel.py* for details, and see "Example Runs" section for result.



9 Run Model

9.1 Example Runs

We have 4 demo runs. Here is the YouTube link: Traffic Vision Demo[39]

9.2 Run This Project On Your Device

If you wish to run this project on your computer, it may take a few hours to setup the environment. Here is how:

1. Python
   Python 3.5 and higher is required to run this project. Besides, Anaconda[34] or just pip[35] is recommended.

2. openCV
   run

   ```
   $ pip install opencv-python
   ```

3. TensorFlow, CUDA and cuDNN
   TensorFlow 2.1 and higher is required to run this project. An TensorFlow-compatible GPU is highly recommended to get expected performance. Before installation, please refer to the chart at button of the page: Tested build configurations[36]. Make sure your

(a) TensorFlow version

(b) Python version

(c) cuDNN version

(d) CUDA version

are compatible with each other. It won't work on GPU otherwise. To be specific, TensorFlow-gpu 2.3.0, CUDA 10.1 and cuDNN 7.6 are recommended.

```
$ pip install tensorflow-gpu==2.3.0
```

Download cuDNN at cuDNN Archive[37], and install CUDA at CUDA Toolkit[38].

4. numpy
   numpy 1.16.0 is recommended for this project. Other versions come out with several errors.

   ```
   $ pip install numpy==1.16.0
   ```

5. Object-detection API
   Git Clone
   https://github.com/tensorflow/models/
   Then, follow instructions on Object Detection API with TensorFlow 2[1].

6. Run Modify *CameraRunModel.py* at line#109 to meet your input source. See opencv-python[31] for instructions. Default input source is YouTube URL.

10 Known Issues

(a) Detections under night (dark) environment are weak
We will figure out a way to balance bright and dark data sizes.

(b) Some Classes are obviously over-fitted
Some classes, for example, *pedestrian* and *Go*, returns relatively high detection scores, even on wrong objects. We will try to add some dropout layers and decrease the sample sizes to see if we can get this fixed.

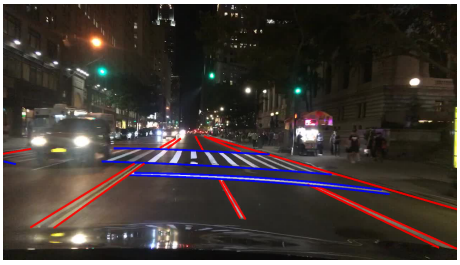
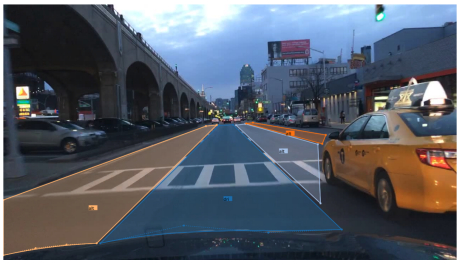
(c) Some Classes are obviously under-fitted
Some classes, for example, *Slow Down* and *Stop*, returns relatively small detection scores. We will try to train the full dataset with 700000 samples of each class to see if we can get this better.

11 Future Works

7. Train full dataset
   We are planning on renting a multi-GPU server to full dataset with 700000 samples of each class. Using two Quadro RTX 6000 GPUs, it will take 105 hours to finish 2 epochs. Cost at about $200.

8. Detect road lanes
   We are planning on creating another model to recognize lanes including white, yellow, solid and dashed lanes. The left figure shows BDD's example of lanes detection.

9. Detect driveable area
   We are planning on creating another model to segment driveable area. The left figure[22] shows BDD's example of lane marks detection. The right figure[22] shows BDD's example of driveable area segmentation.

References

[1] TensorFlow Object Detection API,
https://github.com/tensorflow/models/tree/master/research/object_detection

[2] TensorFlow,
https://www.tensorflow.org/

[3] Your GPU Compute Capability,
https://developer.nvidia.com/cuda-gpus

[4] Waymo began as the Google Self-Driving Car Project in 2009,
https://waymo.com/

[5] Details of Waymo,
https://waymo.com/

[6] NIO Pilot,
https://www.nio.com/news/new-nio-pilot-option-packs-and-navigation-pilot-2020

[7] XiaoPeng XPILOT,
https://www.xiaopeng.com/g3/xpilot.html

[8] Tesla Autopilot,
https://www.tesla.com/autopilot

[9] General Motors Super Cruise,
https://www.cadillac.com/world-of-cadillac/innovation/super-cruise

[10] Waymo is opening its fully driverless service to the general public in Phoenix,
https://blog.waymo.com/2020/10/waymo-is-opening-its-fully-driverless.html

[11] YouTube video *Model 3 - Beta FSD - Test run to Home Depot - 2020.44.10.2 - 14 Nov 2020*,
https://www.youtube.com/watch?v=y0Ft1Xu8BNM

[12] Google Images, https://images.google.com/

[13] iOS quickstart,
https://www.tensorflow.org/lite/guide/ios

[14] GitHub *labelImg*,
https://github.com/tzutalin/labelImg

[15] ASTYX HIRES2019 DATASET,
https://www.astyx.com/development/astyx-hires2019-dataset.html

[16] LYFT LEVEL 5 OPEN DATA,
https://self-driving.lyft.com/level5/data/?source=post_page

[17] nuScenes By Motional,
https://www.nuscenes.org/

[18] Announcing Open Images V5 and the ICCV 2019 Open Images Challenge,
https://ai.googleblog.com/2019/05/announcing-open-images-v5-and-iccv-2019.html

[19] OXFORD ROBOTCAR DATASET,
https://robotcar-dataset.robots.ox.ac.uk/

[20] PandaSet,
https://scale.com/open-datasets/pandaset

[21] Waymo Open Dataset,
https://waymo.com/open/

[22] Berkeley DeepDrive,
https://bdd-data.berkeley.edu/

[23] TensorFlow 2 Detection Model Zoo,
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

[24] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., . . . Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. ArXiv, 1-9.

[25] Roy, P., Ghosh, S., Bhattacharya, S., Pal, U., S., & I. (2019). Effects of Degradations on Deep Neural Network Architectures.

[26] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2016.90

[27] Download SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50),
http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet50_v1_fpn_1024x1024_coco17_tpu-8.tar.gz

[28] IOS image detection example,
https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/ios

[29] Details of tf.lite.TFLiteConverter,
https://www.tensorflow.org/api_docs/python/tf/lite/TFLiteConverter

[30] GitHub issue *SSD ResNet from model zoo not working after conversion to TFLite*,
https://github.com/tensorflow/models/issues/9287

[31] Description of OpenCV,
https://pypi.org/project/opencv-python/

[32] YouTube video *Daily Observations and Trolls of the Week 92 * Subscribers Edition * [Dashcam Europe]*,
https://www.youtube.com/watch?v=hY-KY76kfkI

[33] YouTube video *Wrong-way driver hits a semi and 3 other cars* at,
https://www.youtube.com/watch?v=W0D2GYFCDI4

[34] Details of Anaconda,
https://www.anaconda.com/

[35] Description of pip,
https://pypi.org/project/pip/

[36] see the chart at button of the page about compatibilities,
https://www.tensorflow.org/install/source#gpu

[37] Download cuDNN,
https://developer.nvidia.com/rdp/cudnn-archive

[38] Download CUDA toolkit,
https://developer.nvidia.com/cuda-toolkit

[39] YouTube video *Traffic Vision (Ver 0.1) Demo - Based On TensorFlow Object Detection API*,
https://www.youtube.com/watch?v=rJBkfOtRbzY&feature=youtu.be

[40] Single Shot Multibox Detecter,
https://arxiv.org/abs/1512.02325