CS842 Final Project Report

on

C2TSR: Concurrent Canada-based Traffic Signposts Recognition System

Suby Singh

(ssz389@uregina.ca)

Department of Computer Science

University of Regina, SK, Canada

19 Aug, 2020

Contents

1 Introduction

Road infrastructure has traffic signs and signals as an integral part of it. These signs warns and instructs information to the vehicle drivers and pedestrians using symbols and words. These road facilities help road users to adjust their driving or walking behavior by providing them critical information and sometimes compelling recommendations. These are to ensure that they follow road regulation that is currently being enforced. Without these useful and meaningful signs i.e. if no feedback available on the speed limit, or is not directed about work in progress, turns, or school zone ahead, we might get exposed to more accidents, collisions, and traffic jams.

Car manufacturing companies, such as BMW, are proactively interested in the Advanced Driver Assistance System (ADAS). This is a result of advancement in automotive intelligent techniques. Systems like ADAS have a central controlling activity which not only includes lane detection but also includes traffic signs and signals recognition [8]. These autonomous vehicles must also abide by the legislation of road traffic signs and henceforth these car systems must understand and recognize these rules. To support ADAS and other techniques, I have built a model called C2TSR. C2TSR as abbreviated for Concurrent Canada-based Traffic Signposts Recognition is a deep learning-based model to detect and classify real-time traffic signs and signals. This model has been trained using Canadian traffic signs and signals. So, it's application is specific to Canada or North America region. The model of recognition technique can be incorporated with the ADAS in the autonomous car driving system in real-time scenarios. Pedestrians must also adhere to traffic rules and regulations in order to avoid road accidents. I believe that this model would be advantageous for visually impaired people if it is incorporated with advanced smart eyeglasses which have camera lenses to capture the real-time view as they walk and use the road facilities. Below are some motivational factors that have been considered while picking up this topic:

1. A prelude of fully automated driving or providing semi-automotive driving systems to assist maintaining speed limit and following other regulations
2. Assistant for drivers to be more attentive (if used with speaker mechanism)
3. Reducing the proximity of accidents caused due to driver's distraction
4. Assisting visually impaired individual when incorporated with advanced smart eyeglasses and speaker mechanism
5. Implementing Car dashboard for the indication of the speed limit and other indicators for the comfort of the driver
6. Help in regulating the traffic by informing condition of the road ahead
7. Assist in improving the comfort and safety of the driver on the road
8. All the above points with the aim to achieve zero accident
9. Less number of research/analysis available for Canadian traffic signs due to lack of public dataset

2 Problem Statement

Around 94% of van crashes are caused due to human error as stated by National Highway Traffic Safety Administration(NHTSA) as well as [12][13]. Among these, recognition and decision errors are the most common and frequent reason for accidents. Hence, implementing a technique to avoid accidents is strongly recommended. Nowadays, Vehicles with ADAS in few areas such as automatic braking, lane detection and departure warning system, automatic parking, driver drowsiness detection have been developed [8]. Car manufacturing company Tesla has worked on stop sign and traffic lights but it does not consider all types of signs. And, also this feature has it's own limits as Tesla made cars get slow down after detecting any type of traffic lights including green light [16]. For ADAS or legally blind people using smart eyeglasses, we required to have a model to be actively working on detecting traffic signs and signals to avoid traffic accidents.

Traditionally, many methods including Machine learning[14] and image processing techniques[15] were employed to detect and recognize traffic signs. These have limitations in terms of speed and programmer needs to list the features that need to be learnt by the model. Also, these existing models cannot be used in Canada as these models use different traffic signs. There isn't much work done for Canadian traffic signs due to the unavailability of required dataset. Analysis has been performed on Germany and Swedish, or othe European countries, but their traffic notation is different from Canada as shown in Fig. 1. Hence, considering these points I proposed an application

Figure 2.1: Speed limit sign in Europe (Left) and Canada (Right)

empowering a Canada-based traffic dataset. The proposed model has been built by taking advantage of deep learning method and transfer learning using YOLO i.e. You Only Look Once algorithm[2]. This Canada-based model detects and classifies the traffic signs and signals in real-time scenario. Goal of this project is to contribute efforts in the field of traffic signposts in order to make roads safer place for pedestrians as well as vans. The proposed model can be incorporated with speakers to keep the drivers informed and avoid any distractions which probably be caused by sleepiness or drowsiness.

3 Solution Overview

Based on the report given by [12][13], ADAS technology with its great feature can potentially help in avoiding 36% of the car crashes and 35% of the collision injuries. The C2TSR model can be a great contribution to ADAS technology. This model would help to inform drivers about upcoming traffic signs and warn them about the upcoming situations while they drive vehicles on roads. Basic building architecture of C2TSR model included following commencing steps:

1. Create a dataset of Canadian traffic signs and road signals by extracting frames from the recorded videos using Python and OpenCV platform. images or the raw videos must cover traffic lights and signs during day and night timings as well as climate conditions such as rainy or winter season. As this experiment has been carried out in summer semester, so I could only capture videos in rainy and sunny climate conditions.

2. Build a model using deep learning method (YOLO- You Only Look Once) to classify and detect the traffic signs and signals in real-time scenario

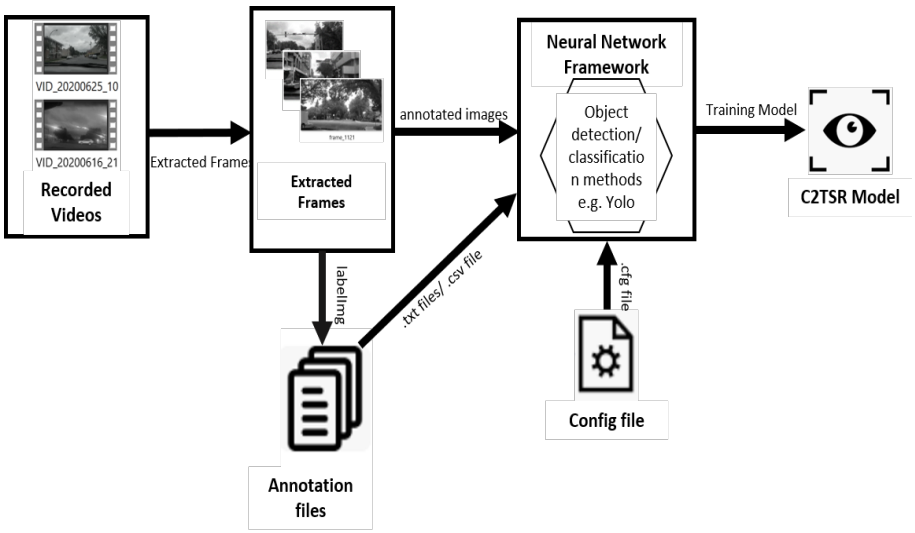


Figure 3.1: C2TSR Model architecture

As shown in the Figure 3.1, we can observe that frames from the recorded videos are being extracted. I have recorded these videos on the streets of Regina while covering different signposts used in Canada. The extracted frames from the recorded videos are then used for creating annotation files. Please refer Creating dataset section for more details on annotation files. The more images or frames and their corresponding annotation files we use, the more accurate our model would be. Keeping this in mind, I have collected around 3,500 frames from the videos and followed data augmentation techniques to create a huge and useful dataset. Creating an useful dataset is a main focus in project.

Later, annotated files and frames/images are used in training YOLO algorithm in Darknet framework. YOLO is a state-of-the-art model for real-time object detection system and it provides faster processing than other algorithms such as Single Shot Detector (SSD) and Faster Region-based Convolutional Neural Networks (R-CNN). Darknet is an open source neural network framework written C. This requires CUDA-enabled GPU card for building darknet and training objection detection model in darknet framework[2]. There are other frameworks too, but darknet with Yolo provides better performance as it is written in C [2]. The generated weight file i.e. trained C2TSR model along with configuration file can be used to detect traffic signposts in images and videos.

4 data

For training C2TSR model, an image dataset has been created which includes images and their corresponding annotation files. Annotation files entail about where and what traffic signs and signals need to be trained for. These annotaion files have been created using labelimg tool and boundary has been marked in rectangular shape which is suitable for this project. Initially, around 5000 frames were extracted from recorded videos. out of these, 3,500 are found suitable to be used for training of the model and rest others were removed because they did not contain any traffic signs or were blurred. Data augmentation techniques such as blurring out the images to bearable extent, increasing/decreasing brightness and contrast were applied on these 3,500 frames. Almost 10 different versions of each frames were generated using above mentioned techniques. This resulted in dataset of around size 3.5 GB. This includes annotation file of each of the frames. for different versions of original images, annotation files are just copied from the original annotation files using a small python script. Following are the examples of such images and their annotation files.

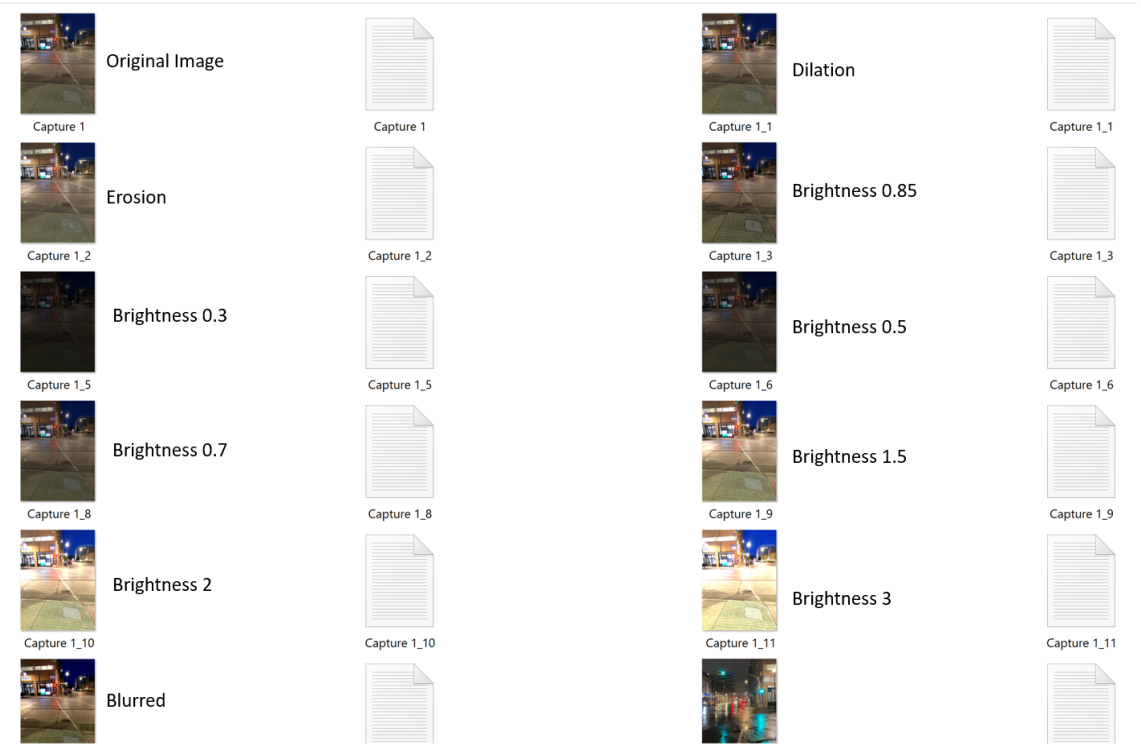


Figure 4.1: An example of a original frame and its various versions

5 Tools

Following is the list of tools that are used while working on this project:

5.1 Annotation

- **LabelImg :** It is a graphical tool for creating annotation files. It let us to create annotation using bounding boxes in rectangular shape. For this project, I have referred https://github.com/tzutalin/labelImg and saved annotation files in .txt format.

5.2 Modules

- **OpenCV-python :** This is a python library used for solving computer vision problems by applying image processing techniques. I have used it for frame extraction, data augmentation and testing C2TSR model on images and videos.
- **Numpy, Panda and Matplotlib :** Numpy library is required by opencv-python as each frame or image is treated as an array. Panda has been used to create a CSV file that lists all the images and it's corresponding annotation details. This csv file is used to visualize the number of samples for each signpost available in the dataset using matplotlib library.
- **Os, Re and Glob :** These python library have been used to iterate through the images available in different folders.
- **Tkinter, datetime, filetype, mimetype :** I have used these libraries to build Graphical User Interface.

5.3 Framework

- **Darknet :** This framework has been employed to train C2TSR model and it provides good performance for real-time object detection as it is written in C[2].

5.4 Platform

- **Jupyter lab :** Jupyter lab is a web browser based IDE for writing python scripts.
- **Google Colab :** Google Colab is an online VM platform provided by Google. I have used it for training C2TSR model.
- **Overleaf :** It is an online tool for writing documents.

5.5 Storage

- **Google Drive :** Created a zip file of the images and annotation files and stored it on Google drive.

6 Prototype of the proposed model

Figure 6.1 depicts the detailed step-wise implementation of the C2TSR model. A test model following this approach is stored on Goggle drive at
`https://drive.google.com/file/d/1FWsZFo-hri63XWJ4bYoNuwXOZDvYbKG7/view?usp=sharing`

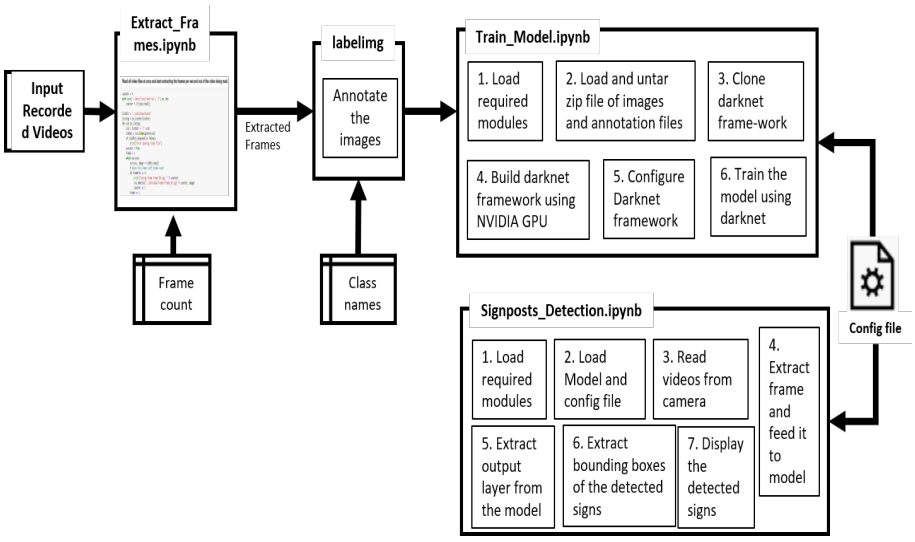


Figure 6.1: Prototype of the proposed model

Firstly to load zip file into the Google colab, google drive is mounted in the VM. Then, zip file is untarred and extracted images and annotation files are stored in darknet folder. Darknet framework is cloned from github repository https://github.com/AlexeyAB/darknet and then built using free GPU provided by Google Colab. In this project, 57 different signposts have been consisdered to train our model. These signposts form total of 57 classes. Configuration file has been adjusted according to these 57 classes. We will discuss more about this in Model building phase. Once the data files and configuration files are ready, we use darknet framework to train our C2TSR model.

7 Timeline

Estimated about 2 months of timeline to work on this project. Following are the major tasks have been considered.

Task	From date	To date
Collect Videos	Jun 15	Jun 25
Extract Frames and Data augmentation	Jun 27	Jun 27
Create annotation Files	July 02	July 22
Train test model using original images	July 22	Aug 01
Train model using all images	Aug 01	Aug 15
Test generated model	Aug 01	Aug 17
Model evaluation	Aug 01	Aug 17
Documentation	Aug 06	Aug 18

8 Expected Outcome

- An User Interface window to load the videos from a folder or camera and display the information on it as it detects and classifies different traffic signs. This graphical user interface will allow users to upload video/images from a folder, or capture it from camera. The look and feel of this is shown on below diagram:

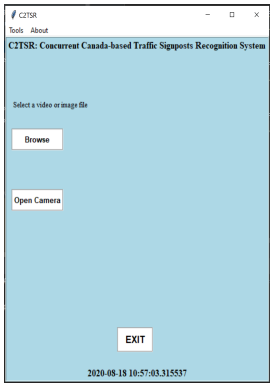


Figure 8.1: Graphical User Interface of C2TSR application

Here is an example of detected traffic signposts on an image by C2TSR model.

Figure 8.2: Example 1

9 Potential Application of the Model

- The proposed model can be integrated with ADAS technology and contribute as a great feature in modern cars. There are many Autonomous vehicle companies such as Tesla, GM Motors, Yandex and Wayo to name a few are involved in developing self driving cars. By CRM staff,

more than half of the drivers in Ontario have no or little knowledge about ADAS as per a survey conducted by Ipsos (a market research and consulting firm) having 1,202 Canadians as participants, in the year 2019 [9]. At the same time another survey conducted by McKinsey & company(a consulting company) with 4,500 participants revealed that most of these were not aware of ADAS features and among those who knew about ADAS, 87% to 89% drivers have bought ADAS equipped cars [10].

- So, we can estimate that as car consumers awareness about ADAS features increases, there will be a potential increase in the number of consumers using these features.
- The proposed model may also be advantageous for visually impaired people if the it is incorporated with advanced smart eyeglasses having camera lenses to capture the real-time videos and keep them giving a sense of warning about upcoming situations while accessing road facilities

10 Business value of the model

- This model will underline a key advancement in the field of autonomous vehicles and their development in ADAS system in Canada and around the world.
- As per [13], it is estimated that market value for Autonomous vehicles will reach to $\$77 Billion$ by 2035.
- There have been some underlying growing demands in the number of workers for the development like these. In 2016, it was seen 213,300 workers in Canada's connected and autonomous vehicle(CAV) where as it is expected to have 3% growth in the numbers of workers i.e. near about 248,000 workers in 2021 [13].

11 C2TSR-Data analytics lifecycle

Lifecycle of C2TSR model follows all steps of data analytics lifecycle phases i.e. Discovery, Data Preparation, Model Planning, Model Building, Communicate results, Operationalize. Following figure depicts all different stages involved in data science-based product lifecycle.

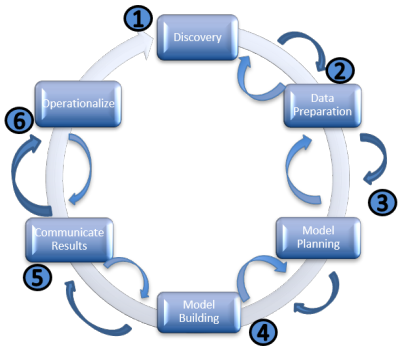


Figure 11.1: C2TSR-Data analytics lifecycle

11.1 Discovery

I started with exploring different domains for finding a suitable problem that can be solved by Data Science. I looked into medical domain and eventually came across road safety issues which interested me. I realized that existing proposed models to solve this issue have been trained using German traffic rules as its dataset is publicly available. I digged more into this and found that car manufacturing companies are also working towards this i.e. a traffic signs recognition feature in ADAS. While doing research in this section, many things came into the pictures. For example, no dataset available to cover north America or Canada traffic signs. So, I decided to continue with this topic i.e. to create an image dataset which would cover most commonly used traffic signs and signals in Canada. Later, build a detection and classification model using this newly created dataset and deep learning technique.

11.2 Data Preparation

I created a new image dataset based on Canadian traffic signs and signals for training C2TSR model. For this, videos capturing the view where I could find the traffic signs and signals are recorded using Oneplus 7T and iphone 11 pro. I have recorded almost 10 videos of varying time duration. Few of them are 8-10 minutes long, few within 5 minutes where as one video of 17 minutes long. These videos cover different lighting conditions and climates. Later, I wrote a python program to perform following steps:

1. Read the videos from folder
2. Extract one frame per second from the videos
3. Save it in a folder

Followings are a few details of dataset created for this project:

- **Source:** Extracted frames from recorded Videos
- **Format:** JPEG images with different brightness and contrast level, and an annotation file corresponding to each extracted frame from the recorded video
- **Access:** Google drive and used Google Colab to access it
- **Statistics:** 50+ different classes of traffic signs and signals
- **Risk factors while preparing dataset :**
  - Different lightning and climate conditions
  - Poor visibility
  - Camera alignment and focal point
  - Motion effects while capturing video
  - Different traffic colors for signs and boards
  - Damaged traffic sign boards

11.2.1 Creating Dataset

A frame per second from the recorded video is extracted using openCV and python. Initially, the extracted frames are stored on local machine. Once all the required frames are extracted, I created annotation for all the traffic signs present in a frame using LabelImg tool. LabelImg is an annotation tool written in qt and python for Graphical User Interface. It generates annotation in XML/PASCAL VOC or yolo format [4]. For this project, I created bounding boxes for the traffic signs in yolo format. It is a simple format that contains class id, coordinates (x and y), width and height of the rectangular portions which are manually drawn to label different objects in an image. It creates a simple text file having annotation details at each line per object in the corresponding image [4].

- **Step 1:** Record Videos and collect images via web scrapping
- **Step 2:** Extract frames (1 frame per second) and remove unwanted frames
- **Step 3:** Create class/data file. Class Id starts from 0. Following is the class/data file created for C2TSR model:

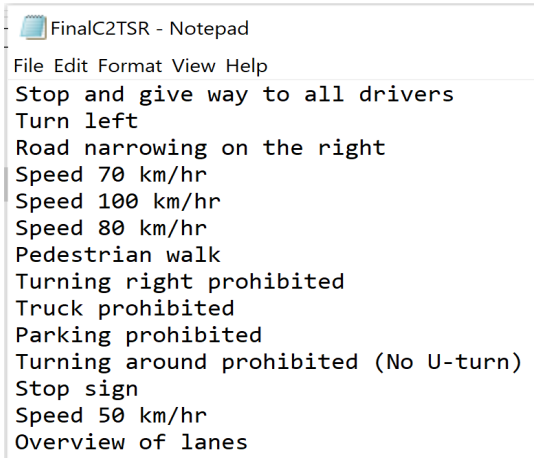
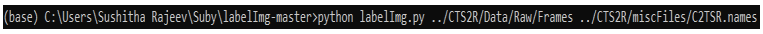


Figure 11.2: Class names

- **Step 4:** Create annotation files corresponding to each frame using labeling tool such as labelimg
  Following is the command to run this tool:



Following is an overlook of LabelImg tool while annotating a frame/image in figure 11.3, and later an example of annotation file generated by LabelImg tool is given in figure 11.4:

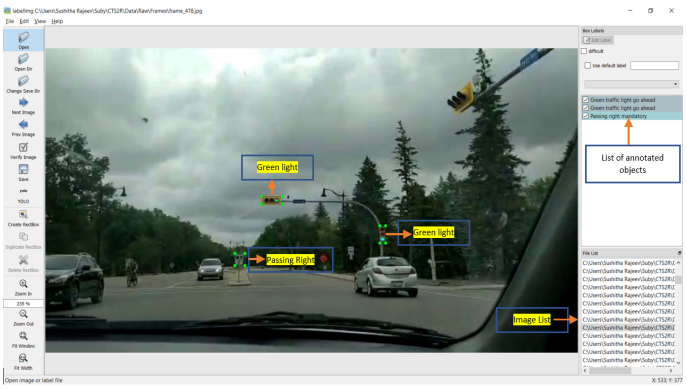


Figure 11.3: LabelImag Example

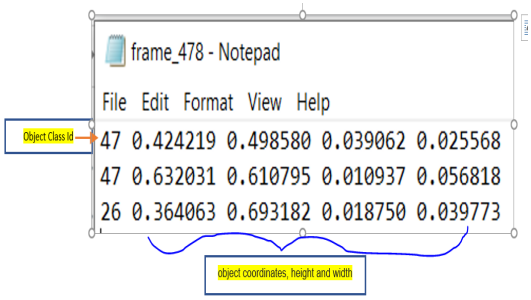


Figure 11.4: LabelImag annotation file

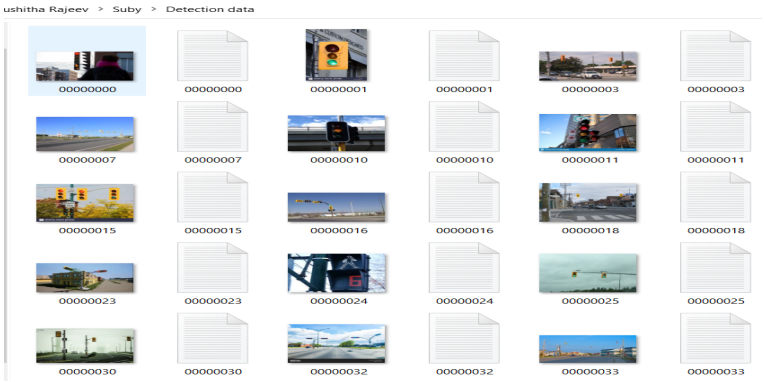


Figure 11.5: Folder having frames and it's corresponding annotation folder

- **Step 5:** Store dataset on Google Drive and access it using Google Colab notebook. Create a zip file of the image for speedy transfer of the files to Google drive. Later, this zip file is loaded into data folder of darknet framework and used while training C2TSR model.

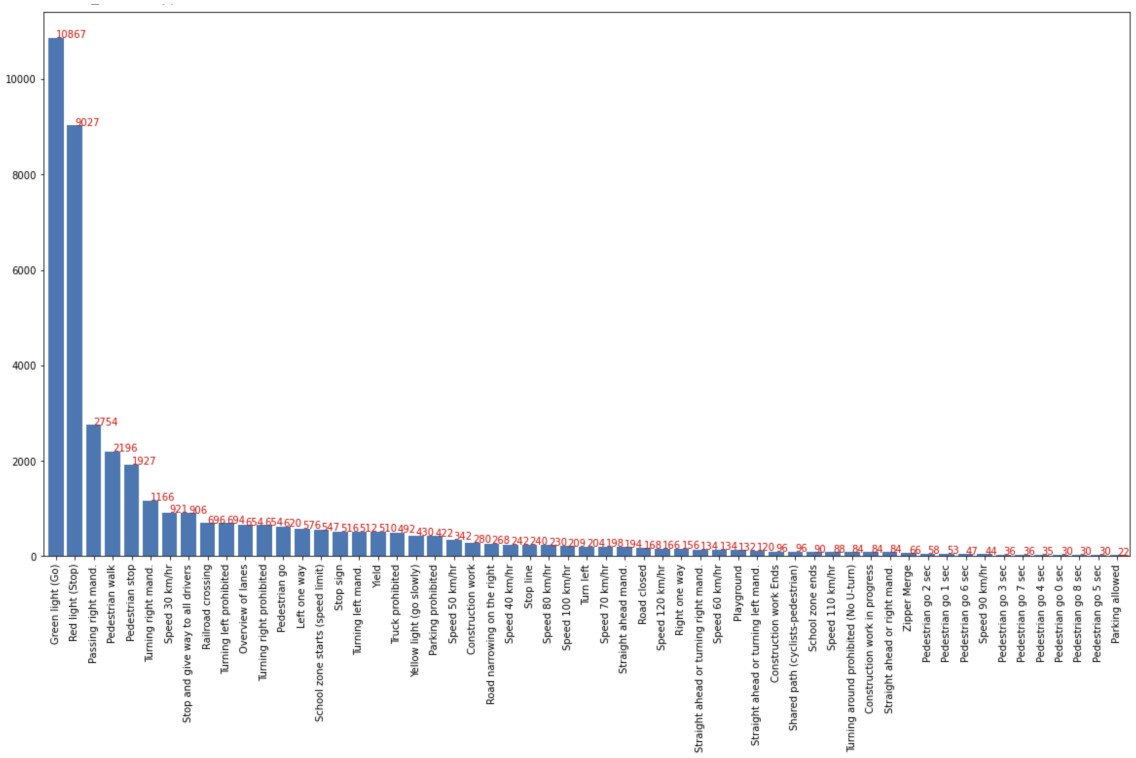


Figure 11.6: number of annotation for each class

11.3 Model Planning

In this phase, due to constraint I decided to train my model using transfer learning. I needed to find out a method that would localize objects and grab activation from the final layer of the convolution. I needed to out a Neural network algorithm that can be suitable for training C2TSR model i.e. fast and accurate neural network. There are many state of the art algorithms are present which can be used for real-time object detection such as Single Shot Detection(SSD), R-CNN, etc. SSD performs tasks such as localization and classification in single shot which means in a single forward pass of the network. I also looked at R-CNN i.e. Region-based Fully-Convolution Neural Networks. In this method, detection is performed in two steps, first that performs selective search to extract regions and second step identifies a number of bounding-box object region candidates. CNN in this method extracts features from each region and feeds them to SVM which further predicts the presence of an object in a particular region candidates. One of the major drawback of R-CNN is that it is very time consuming. To overcome time problem with R-CNN different versions of R-CNN were developed i.e. Fast R-CNN and Faster R-CNN. Faster R-CNN is much faster than R-CNN and Fast R-CNN. All these methods look at the region of the image rather than the whole image at

once. This means that part of the image is looked at. I explored further and found YOLO i.e. You Only Look Once to be much faster and accurate than other algorithms [4] (Please refer below figure for comparison of these algorithms.

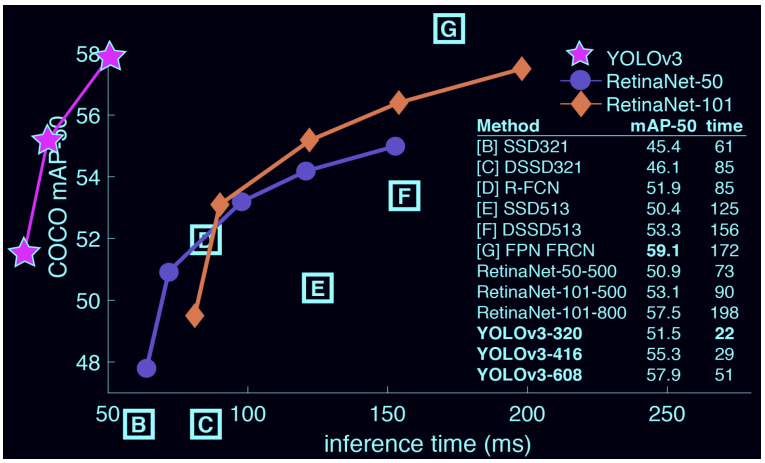


Figure 11.7: comparison of YOLO V3, source [4]

YOLO is much different from region-based algorithms. It looks at the image once and detects the boundary boxes, and classifies the objects in a single convolution layer. Classification is done based on the probabilities of having each class. Based on accuracy and speed criteria, I decided to go with YOLO v3 deep learning real-time object detection method.

11.4 Model building

After finalizing the method by which C2TSR to be trained i.e. YOLO V3, I started working on its requirement. YOLO is trained on Cocoa dataset and all the files and parameters are set according to it. So, modification to these files as per our model is required. I decided to use darkenet framework to train the model. Darknet is written in C and much faster than other framework such as pytorch. I cloned darknet from https://github.com/AlexeyAB/darknet and built it using GPU. Later, I worked on configuration (.cfg), data (.data), and names (.names) files. Names file is same as the class file created while building the dataset. It contains name of the classes to be detected. Data file contains information such as training and testing datset, number of classes, and path to store the model and .names file. Following figure shows the content in .data file:

```
classes = 57
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = /mydrive/trainC2TSR/latest/
```

Figure 11.8: data file

11.4.1 Adjusting configuration file

Darknet trains the model by looking at the parameters' value as mentioned in the configuration file. YOLO configuration file has multiple such parameters like learning rate, angle, steps, batches, number of classes, number of filters, etc.

```
[net]
# Testing
batch=64
subdivisions=16
# Training
# batch=64
# subdivisions=16
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 114000
policy=steps
steps=91200,102000
scales=.1,.1
```

Figure 11.9: YOLO configuration file snippet

These parameters need to be set according to the model to be trained. For C2TSR model, I have configured following parameters:

- **batch = 64** (this many images+labels are used in the forward pass to compute a gradient and update the weights via backpropagation.)
- **subdivisions = 16** The batch is subdivided into this many "blocks". The images of a block are ran in parallel on the gpu.
- **max_batches = 114000** (2000 multiplied by # of classes)
- **learning rate = 0.001**
- **steps = 91200 ,102000** (80% of max_batches and 90% of max_batches respectively)
- **scales=0.1,0.2** adjust learning rate i.e. multiply the learning rate by 0.1 after 91200 steps, then after 102000 multiply it again by 0.2
- **classes = 57**
- **filters = 186** (# of classes + 5) multiplied by 3 - # of convolutional kernels in a layer
- **angle**, in degree; augment images by rotation up to this angle layers

- **activation** Activation function, relu, leaky relu, etc.

Once all the required files are ready, I copied these files to darknet/data folder along with the images and started training C2TSR model. darknet53.conv.74 is a pre-trained model.

```
[ ]  # train your custom detector
     !./darknet detector train data/obj.data cfg/yolov3_custom.cfg darknet53.conv.74 -dont_show
```

Figure 11.10: command to start training the C2TSR model

11.5 Communicate Results

After training for 10 continuous days, I stopped training the model at **80,000** iterations where the average loss obtained is around **0.14**. Following the graph which shows dropout of average loss at 1000 iterations while training C2TSR model.

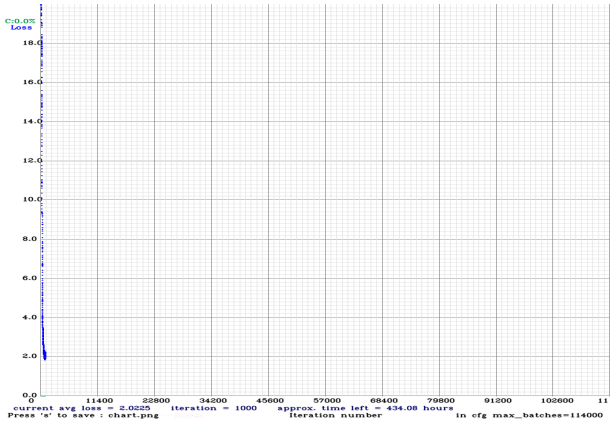


Figure 11.11: Average loss dropout graph at iteration 1000

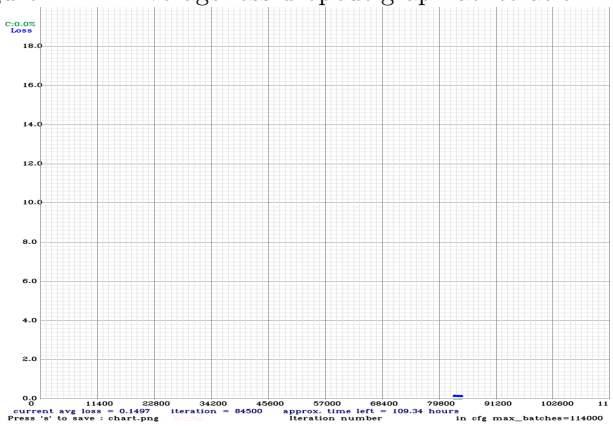


Figure 11.12: Average loss dropout graph at iteration 80000

These graphs are generated by darknet while trainig the model at every 100 iteration.

The generated model is first tested using a set of images to detect the traffic signposts. Criteria for measuring the success is accuracy and confidence level for the detected objects in an image. These values are shown alongside the detected class names. Once successful, the model is then be tested using the recorded videos or videos taken from the Camera. While testing on videos, frames per second is extracted and same method as used for images is applied on each frame. The boundary boxes and accuracy level is collected from the output layer of the CNN. I have created an User interface window to load the videos from a folder or camera and display the information on it as it detects and classifies different traffic signs.

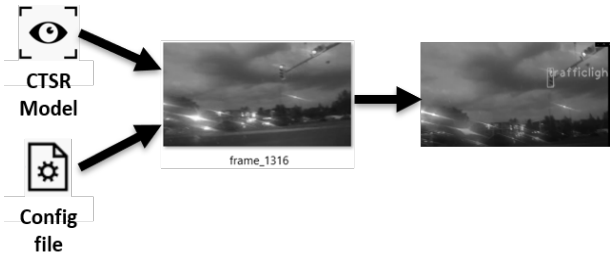


Figure 11.13: C2TSR model testing using images

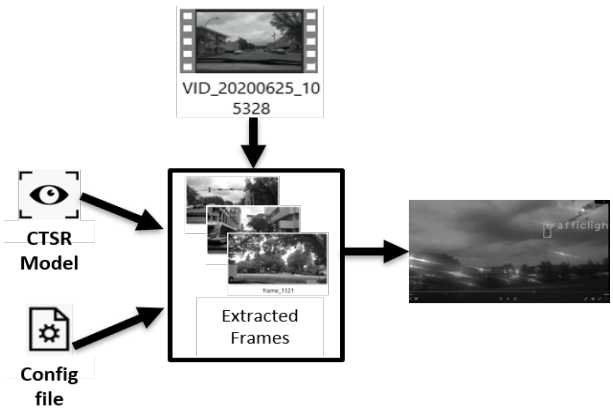


Figure 11.14: C2TSR model testing using videos

I have uploaded a video on YouTube which shows the detected traffic signs and signals. Please click on below links:
`https://youtu.be/l7xEClga5yI` Here, I have uploaded to output of c2TSR model testing. This video has been created as the combination of different videos to cover different light conditions.

Code snippet to test the C2TSR model:

```
#loading model and configuration files
net = cv2.dnn.readNet("../Model/yolov3_custom_last.weights", "../miscFiles/yolov3_custom.cfg")
#get layers
layer_names = net.getLayerNames()
#get output layer
output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
colors = numpy.random.uniform(0, 255, size=(len(classes), 3))
#input video
cap = cv2.VideoCapture(file_path)
#set font of the text
font = cv2.FONT_HERSHEY_COMPLEX_SMALL
starting_time = time.time()
frame_id = 0
while True:
    #read frames
    _, frame = cap.read()
    frame = cv2.resize(frame, None, fx=0.4, fy=0.4)
    frame_id += 1
    height, width, channels = frame.shape
    # Detecting objects
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
```

Figure 11.15: C2TSR model testing - code snippet

11.6 Operationalize

Ideally C2TSR model should be deployed in ADAS system on wide scale production environment. For this project, I created an application which runs this model in background. The developed application allows either to select input video and image from the local computer or take input from device camera. Following is an overview of the application.

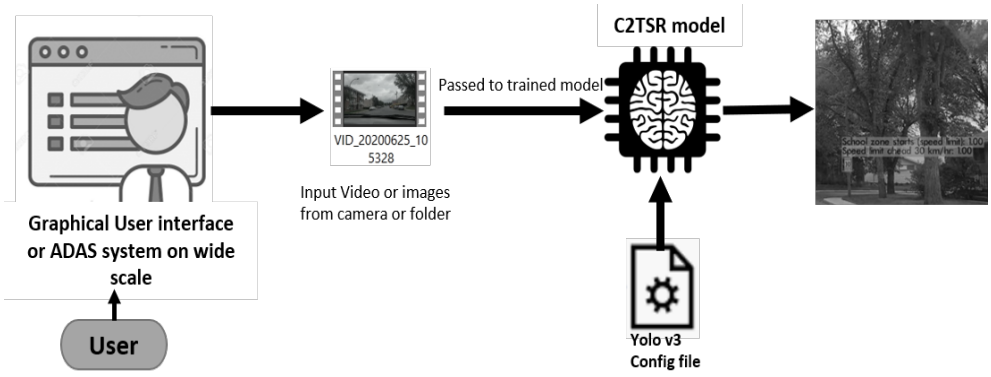


Figure 11.16: Process of using C2TSR model

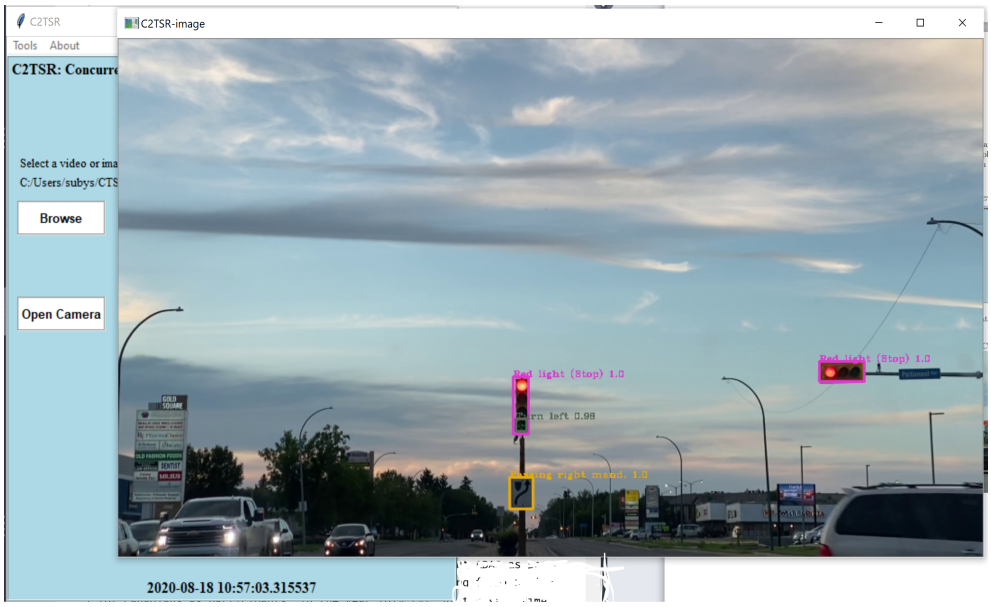


Figure 11.17: Application screenshot

`https://youtu.be/jlay6j7ZRp4` **Here, I have attached a recorded video to showcase how C2TSR model can be used and deployed.**

11.6.1 Future work

Due to time constraint, I managed to cover as many traffic signs as I could collect. There are many different traffic signs left out that need to be collected and updated in the dataset. Also, this is an imbalanced dataset. For example, there are less number of images used for speed limits. So, one can collect more images and train the model using the last trained weight file.

12 Contributor

1. Suby Singh worked on videos recording, collecting images/frames, creating annotation files, creating dataset, data cleaning, model training, model testing, deploying model using an UI, documenting the project slides, documenting the project proposal and report, and recording the project presentations. She has worked in an IT firm for 5 years and did not have any experience of data science techniques prior to taking this course.

References

[1] Ministry of Transportation, https://www.ontario.ca/document/official-mto-drivers-handbook/signs

[2] Redmon, Joseph and et al., "YOLOv3: An Incremental Improvement", arXiv, 2018

[3] S. Xu, Z. Zhao, X. Wu, "Object Detection with Deep Learning: A Review", IEEE Transactions On Neural Networks And Learning Systems, 2019

[4] Tzutalin. LabelImg, Git code, https://github.com/tzutalin/, 2015

[5] www.AdcIdl.com, International Driver's License Application online, Canadian Road Traffic Signs"

[6] sgi, "2019-20 Saskatchewan Driver's Handbook" https://www.sgi.sk.ca/handbook/-/knowledge_base/drivers/traffic-signs, 2019

[7] GTSRB, https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

[8] Wikipedia, "Advanced driver-assistance systems", "https://en.wikipedia.org/wiki/Advanced_driver-assistance_systems"

[9] CRM staff, "https://www.collisionrepairmag.com/adas/", 2019

[10] McKinsey and company, "https://www.mckinsey.com/industries/ semiconductors/our-insights/advanced-driver-assistance-systems-challenges-and-opportunities-ahead#"

[11] Cutean, A. Autonomous Vehicles and the future of work in Canada, Information and Communications Technology Council(ICTC), Ottawa, Canada, 2017

[12] Canadian Auto Dealer, "https://canadianautodealer.ca/2020/01/canada-is-behind-in-adas-awareness-training/", 2020

[13] Belron Canada Inc., The hidden link between windshields and road safety, https://www.speedyglass.ca/DATA/CMSDOCUMENT/46.pdf, 2019

[14] R. Kulkarni, S. Dhavalikar and S. Bangar, "Traffic Light Detection and Recognition for Self Driving Cars Using Deep Learning," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 2018, pp. 1-4, doi: 10.1109/ICCUBEA.2018.8697819.

[15] C. yu, Y. Bai, A Traffic Light Detection Method, Advanced Technology in Teaching pp 745-751

[16] Kirsten Korosec, https://techcrunch.com/2020/04/27/tesla-vehicles-recognize-and-respond-to-traffic-lights-stop-signs-with-latest-software-update/, 2020