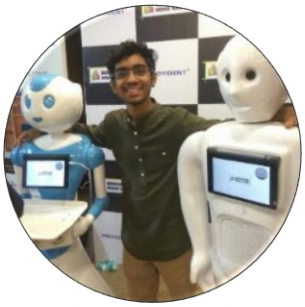
Submission for Summer 2019 Internship Evaluation



Sai Siddartha Maram
Former Intern Invento Robotics, India
Thapar Institute of Engineering and Technology

Part 1: Understanding the dataset

The German Traffic Sign Detection dataset is a dataset comprising of 43 classes, each class contains a collection of distinct images belonging to a particular sign. Our task here is to detect "RED ROUND SIGNS". On studying the Readme from the downloaded dataset, we identify only certain classes to fit the purpose. The below mentioned classes belong to the category "RED ROUND SIGNS"

Classes: [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 16] (Classes with red round signs)

Now, the dataset is partitioned into two segments train and test. The train dataset contains a bunch of images for each class, while the test dataset contains a set of images which are not partitioned by class.

1.a Getting some real images (I would prefer to see the images)

It is evident from the download the images are mentioned in the ". ppm" format, which is hard to visualize. For the better understanding of the images, it would be beneficial if we could see them in the form of JPEG. There are two methods we can approach this,

a. Use on an online converter (take one .ppm at a time and convert to .jpeg)

Pros: Nothing to code

Cons: Since we have above 500 training images, this is going to take a lot of time

b. Write some code to convert all the images to .jpeg

Pros: Quick

Cons: None

The mentioned link contains a written code snippet which can be used for image conversion. The code uses the PILLOW library from python which helps us with a bunch of operations on images using python.

1.b Decoding and understanding the ground truth file

Consider any one line of the ground truth file 00000.ppm;774;411;815;446;11

- 00000.ppm : refers to the image file the traffic sign is located in.
- Field 2 to 5 describe the region of interest (ROI) in that image.
- Here Xmax: 774
- Here Ymax: 441
- Here Xmin: 815
- Here Ymin: 446
- ClassID : 11 (Which class the image belongs to)

This file is of extreme use because it **saves us from the burden of drawing bounding boxes** and generating .xml files which in turn need to be converted into csv files to perform object detection (using TensorFlow). To better evaluate my understanding of the ground truth file [Image,(Xmax,Ymax) , (Xmin,Ymin), class] I would strongly recommend to look into GitHub : Understanding the GroundTruth (this link contains code where neat bounding boxes are drawn across the classes of objects).

Filtering images and Ground Truth CSV's only for classes [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 16]
Open the CSV (Training) in excel and filter out all the other classes using excels filter tools. For the images, how I approached to collect all images from the specific classes is as follows, from the sequential numbering of images in the train dataset, choose only those images whose image (filename) exists in the CSV which deals with classes [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 16]. A link to the csv with only required classes is here : [GitHub: CSV required for training].

Note: For training object detection on various architectures, I will be using the TensorFlow object detection for support. The TensorFlow object detection generally has a procedure to initially draw bounding boxes (generate .xml) further convert this into .csv. Since we already have the bounding boxes, what I will be doing is convert the existing CSV into a manner suitable for the TensorFlow object detection and then use it to run different models.

Manipulating the CSV for Training purposes.
The manipulated CSV looks as follows as opposed to the traditional
00000.ppm;774;411;815;446;11 [observe the format]

New CSV

filename	width	height	class	xmin	ymin	xmax	ymax
00003.jpeg	1360	800	redround	742	443	765	466

Points to note:

- To get the size and dimensions of the image use[GitHub: dimensions]
- The biggest manipulation is the class, I have converted all the different classes which appear like red round signs to "redround". [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 16] == redround
- The filename, has been changed to .jpeg format which we obtained in the above process.

So, at the end of this step, we have the following important file structure

a. train_image [train]: A directory containing .jpeg images of the training data from classes [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 16]
b. Test_image [test]: A directory containing .jpeg images (cut and paste about 7-10% of the images from the test_images to train_images)
c. train.csv [CSV files]: A csv folder in the previously mentioned format
d. test.csv : A csv folder in the previously mentioned format

Alternative Datasets:

1. A very close and interesting dataset Belgian Traffic Dataset. I say interesting because of the fact on a mere scroll down through images, to the naked eye are a number of images

where lighting conditions are poor. In a production level scenario, I believe people tend to miss traffic signs during the night and hence detection systems with low lighting would be beneficial.

2. Video frame-based annotation is available in LISA Traffic Sign Dataset, apart from being annotated on videos, I have had no permission to check out the dataset, hence cannot further comment on it.

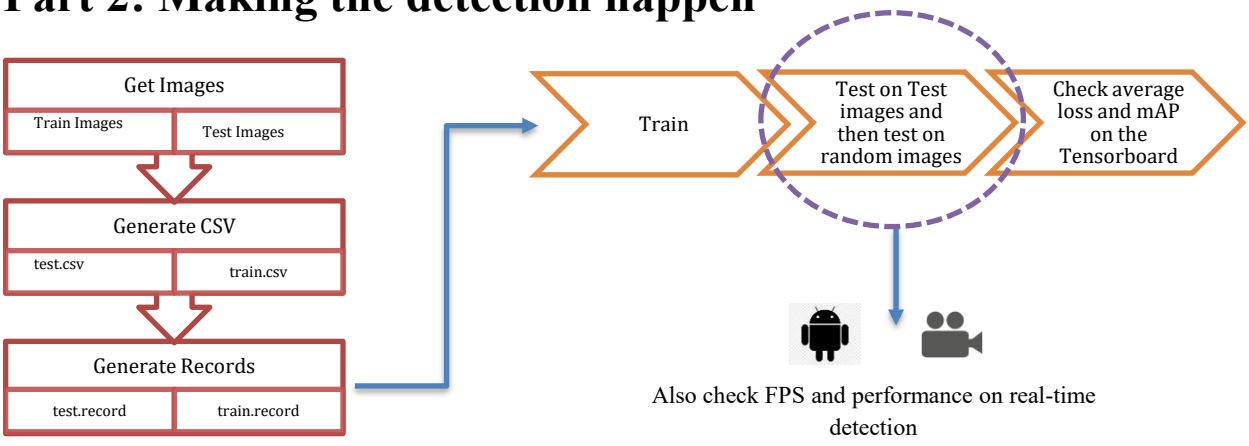
All Weather based Detection

To the naked eye, it is absolutely tough to identify the red round circles (for the below image), this means it's almost a no chance for object detection models, here after some thinking, I realized the entire image is made up of RGB values, if I maybe can add certain factor (reasonable factor) I might be able to brighten up the image. A simple internet search allowed me to realize, this concept is Histogram equalization. On Image 00377.jpeg, which is a cloudy day, an application of histogram equalization makes it much easier to identify the red round circles.

Personal opinion on the Dataset:

Since the dataset has 43 classes and each class has certain images, on opening each folder to have a look at the images, what surprises me is the inconsistency in number of images per class, just in case we further look to perform classification there is a chance that the model might be more biased towards certain classes with a greater number of images.

Part 2: Making the detection happen



2.a Choosing the Neural Network

Before choosing the neural network, it is important to discuss certain objectives which we look to meet using the neural network

1. To detect red round signs on the road
2. Has to be real time (quick)
3. Has to be suitable enough to work on embedded systems / smartphones
4. The region of interest in all the images are really small

I have read the paper corresponding to Yolo [paper] and YoloV2 [paper], it clearly mentions it struggles in the detection of small objects (Objective 4). Even through personal experience on training Yolo during previous internship experiences, Yolo might not be the best neural network model for real time detection of objects **with small screen space.** On the other hand, YoloV3 takes a tradeoff between speed and accuracy and is more accurate on smaller objects. (This conclusion is from intuition and would definitely want to give a try).

For quick real time detection even on embedded systems, a decent neural network would be the **ssd_mobilenet**. I will walk through the slightly tweaked parameters and why I have changed the parameters as per the dataset. (considering the hardware support, I have). I could have gone just with a simple ssd neural network, but to increase the FPS on android phones or embedded systems it is highly recommended to use feature extractors such as mobilenet.

Hyperparameters:

The default configuration file for ssd_mobilenet has a batch size of 24 and runs for a total 20,000 epochs. A batch size of 24 is generally preferred when you are going for a large number of classes.

Let, a→ batch size

Epochs_1 → iterations

Let b → batch size

Epochs_2 → iterations

Can we expect the same level of performance on both the hyperparameters provided a*(epochs_1) == b*(epochs_2)??

In hunt of this question I read the paper (Batch Size). The **abstract** clearly mentions "**With the increase in model there is a degradation in the quality of model**". Since, I am only training on one class and since the ROI of the class is very small, I would prefer the model learns very minute details, hence **I bought down the batch size by half** and left the **learning rate small 0.004**. [On trying with a learning rate of 0.01, the average loss kept oscillating between 0 and 6]. **The optimizers used are RMSprop**. This optimizer is a default inclusion of the ssd_mobilenet which I did not hamper with.

{**Reasons 1.0:** This machine learning problem is definitely not a simple convex curve, using SGD or BGD would land us in some local minima missing the global minima.}

{Reason 2.0: Unlike the other descent algorithms, this is an adaptive algorithm it updates the learning rate over time [formulas mentioned in Geoffrey Hinton class on Coursera]**}**

Generating Record Files

The images and its corresponding .CSV have to be converted into a format which can be fed into TensorFlow object detection. For this purpose we generate train.record and test.record corresponding to test and train respectively. [.record]

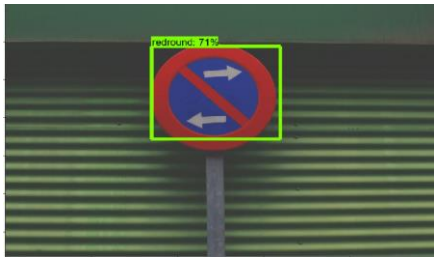
Training

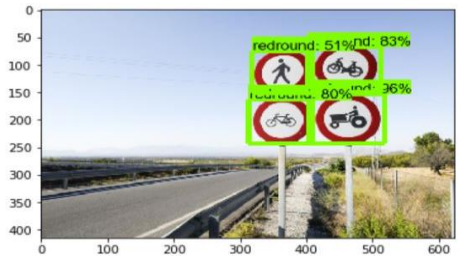
Use the TensorFlow object detection (python train.py) which takes in the record files, csv and let it run for iterations, till the average loss is consistently less than 1. In my case the average loss started of very high at 16 and after 2160 iterations it hit a value of nearly 1.29 which gave quite satisfactory results.

Testing and evaluations:

For evaluating the model, the approach, I used is as follow.

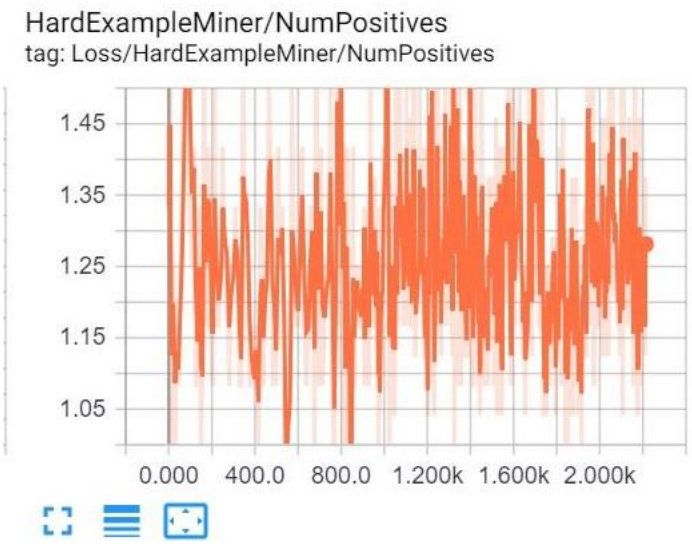
a. Try on the test dataset provided by GTSD
b. Try on totally random images from the internet and check.





c. Then check metrics (average loss), IoU

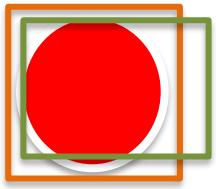
Average Loss:



This graph indicates the average loss [which has to be kept low as possible] after 2160 iteration. The average loss for this model is nearly 1.29, further more iterations might get it near 1 or below. To perform state of the art detection a model is expected to have an average loss of less than 1.

IoU:

The only major concern I see in this model is the IoU on totally random images is not up to state-of-the-art detection models. IoU stands for Intersection over Union a better way to visualize this is through a small example:



Let **Orange** denote the ground truth and **Green** denote the detection, there is certain region which is an overlap and certain region which is a union, **IOU = Area of overlap / Area of Union**. State of the art object detection models such as Yolo achieve an IoU of almost 1. By mere visual intuition it is clear my model achieves IoU greater than 0.7 but less than 1.

Possible Reasons:

- Very few test images, poor IoU happens when a model has been given less number of images to test itself on.
- Small RoI, and no data augmentation. Small screen presence makes it tough to get most of the features, and eventually end up loosing a lot of certain information/features.

Part 3: Envision

Approaching real time detection on Potential Boards on Autonomous car:

Jetson Tx2:

The JetsonTx2 is Nvidia's state of the art GPU board, which is portable and widely used across many autonomous navigating cars. Since it offers a Linux environment a combination of Python's image processing library would allow you to perform real time object detection with TensorFlow's support. link

Approaching real time detection on embedded systems:

For real time detection, I have taken a bit of inspiration from **Anirban Chatterjee's** SmartPhone Road assessment presentation and converted my model smartphone friendly for sign detection. Here are a few screenshots of the detection model performing in real time (video stream). Please feel free to download the .apk file on your smartphone for testing Android App

Embedded AI:

The idea of running state of the art machine learning algorithms on embedded systems was long running in my mind. From my research internship experiences and knowledge some practical approaches to this problem is as follow:

Microsoft Embedded Learning Library:

The .ell model developed by Microsoft allows the smooth running of classification algorithms on the raspberry pi. The accuracies are definitely on the lower side, but provide decent performance.

TensorFlow Lite:

TensorFlow has provided a training strategy where, in place of the regular training the model generated would be a model suitable for embedded systems. [Not given a try, the official documentation of and implementation instructions seem quite convincing]

The Neural Stick: [Intel]

Intel Movidius has to be one of the best bets on performance when it comes to implementing machine learning algorithms on the single board computer. Its cost might be a little on the higher side, but would definitely be worth a try.

Personal Machine Learning Desires / interests / plans :
[note: this section is in response to, "what machine learning algorithms I would like to try and implement"]

A. Applications of GAN's

I am really fascinated by the concept of GAN's not just the research but looking at the potential possibilities it can lead if bought into production. I have had my hands on writing basic GAN's which deal with generation of images from raw signals whose link is[GAN using PyTroch].

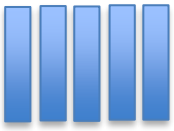
After this I have restricted myself and never spent good time on it due to academic constraints. So, when the opportunity strikes, I would love to further dig into writing GAN's for practical applications and develop a good hands on experience.

B. AI in sports [cricket]

Cricket has been the go-to play time sport for me. I have always been searching for time to develop models for cricket shot prediction based on various cricketing parameters and sensors, the mere fact sensors to detect ball pace, landing are expensive lets me to put the idea back into the shelf.

C. Transfer Learning for Sign classification

I will illustrate what I have in mind through a diagram.



Assume this is pretrained [ImageNet] resnet50, now instead of training the whole neural network from scratch, an interesting approach would be keeping the features it learnt from ImageNet and remove the last layer and write a custom layer for sign classification {transfer learning}.

The orange would indicate the sign detection layer. In terms of code (keras) it would look something like this

```
num_classes = 10 //assume 10 trafic signs
path = 'path_to_pretrained_resnet_weights'
traffic_model = Sequential()
traffic_model.add(ResNet50(include_top=False, pooling='avg', weights=path))
traffic_model.add(Dense(num_classes, activation='softmax'))
```

```
#don’t train the first layer since we load it up from resnet.
traffic_model.layers[0].trainable = False
```

Areas to Improve in the model and potential reasons for loss in accuracy / False Predictions: {Important}

A few points which I should have thought before the training

- **Normalization** is something I believe, I should have done. Basically, all state-of-the-art machine learning models perform image normalization, this is mostly done to treat image uniformly. So, subtract each pixel/image with the mean of the dataset and then use the standard deviation to divide.
- **Optimizers:** RMSprop optimizer is definitely the state-of-the-art optimizer, maybe next time before training I should run a few iterations to evaluate performance on other state of the art optimizers such as AdaMax and Adam, which are also adaptive in nature but have a few additional terms which help in deciding the learning rate.
- **Zooming up images:** Consider the below image, the region where the red bounding box

is present is really small in compared to the image, Keras provides a data augmentation tool, which allows us to zoom up the image and maintain the corresponding ground truth. In terms of code, it would look like below.

```
#this function is generally is applied on a batch of images to augment each
#image in the folder
data_generator_with_aug = ImageDataGenerator(preprocessing_function=preproces
s_input, width_shift_range = 0.2, height_shift_range = 0.2)
```

Classification challenges: [If we are trying to write the NN from scratch]

Data Augmentation:

It is only fair we have a reasonably equal distribution of images across all classes, else there might be a bias in classifying a class more than the other. This can be sorted using data augmentation by performing the following:

a. Rotation
b. Increasing the contrast
c. Flipping
d. Projection
e. Random cropping

f. Color Jittering [change one of the RGB values by some adding or subtracting some random value]

Dropout selection:
1. Should I add the dropout only after the fully connected layer?
2. Should I add the dropout only after the CNN
3. Should I add one after both the CNN and fully connected layer.

These questions need to be answered. In general, the 3rd way mentioned above is preferred, but better conclusions can be only drawn only after running a few epochs.

Careful generation of validation and test dataset:

It might be very tempting to randomly split the dataset, but it is better we sit and pick a few images from different classes based on label name, otherwise there might be information leakage and the accuracies we get on test and validation sets might differ greatly. This is from previous experience assume, I have a dataset with 60 images for each class and if I split the dataset randomly, we get good validation accuracy and poor test accuracy, this indicates the model is of poor design or there has been overfitting in the validation set.