

Vehicle Detection using TensorFlow

Priyam Saikia, *Graduate Student, University of Florida*

Abstract—As we move towards self-driving cars and intelligent traffic systems, it becomes imperative to develop robust as well as fast vehicle detection algorithms. The objective of this project is to train, test and determine the most effective approach to implement vehicle detection using neural networks. This project focuses on two major pre-trained models, Faster Region-based Convolved Neural Networks (Faster R-CNN) and Mobile Single Shot Detectors (Mobile SSD). Both the models are studied, implemented and tested with specific configurations and their results are analyzed and compared. TensorFlow object detection API is utilized to train the models. Custom annotated datasets have also been used to provide better results for vehicles. The results suggest that Faster R-CNN excels in accuracy while SSD stays on top in terms of speed. IBM Watson Machine Learning and cloud annotation tool is also explored at a basic level to analyze on how to harness the computational strength of such powerful cloud AI tools.

Index Terms— Faster R-CNN, Mobile SSD, TensorFlow, Vehicle Detection, COCO, Loss function

I. INTRODUCTION

OBJECT detection has become a popular application of machine learning in the recent years. With the modern world gravitating towards artificial intelligence and day-to-day automation, it becomes essential to develop robust object detection models as well as algorithms. New algorithms with better localization and more efficiency are developed every day by various open source organizations [1]. As a result, researchers and developers can refer to these already developed algorithms and build better models and tools on top of that. This turns out extremely helpful as it saves redundant efforts and paves way for rapid progress in the field of object detection.

In this project, a more specific facet of object detection has been explored – vehicle detection. Vehicle detection has matured very fast in the last decade. As we move towards intelligent traffic systems and self-driving cars, accurate vehicle detection methodologies remain the crux of the problem statement. However, for the vehicle detection tools to be helpful for a diverse set of applications, it is necessary to focus on many related areas that requires analysis using machine learning too.

For instance, traffic monitoring, speed monitoring, vehicle classification, traffic counting or vehicle detection for spying agencies may require the detection of just the vehicles while ignoring the environment around the vehicle. However, if vehicle detection is to be implemented for a self-driving car,

other areas of focus such as lane detection, road sign interpretation as well as obstacle detection comes into picture too. As a human, we constantly keep track of our environment when we drive. Similarly, when we are teaching our vehicles to see, we need to teach them to understand their surrounding environment as well as the impact it has on traffic safety [18].

Another important aspect to keep tabs on while developing vehicle detection tools, is efficiency [10]. While efforts are being made every day to develop effective and state of the art vehicle detection models, it is also imperative to focus on efficiency as we move forward. As more and more industries adopt intelligent automation, it will require us to shift from doing just the right thing to doing the right thing in the best possible way.

This project explores the stand-alone vehicle detection problem. It includes detection of cars, trucks, motorbikes and so on. Two pre-trained models from TensorFlow models would be used for major chunk of this project. The results from the two would be studied and analyzed. As a part of the related work, IBM Watson Cloud Machine Learning tool will also be used to analyze the benefit of powerful cloud annotation and training tools. This will be used with React js and tensorflow js to do a basic detection for moving vehicles.

II. AN OVERVIEW

A. Problem Description

Extensive literature survey on object detection and vehicle detection has been done as a part of this project [10]. It is seen that considerable research has been conducted on the area of deep learning approaches, special using region-based neural networks [2] [28]. Object detection tends to be more difficult

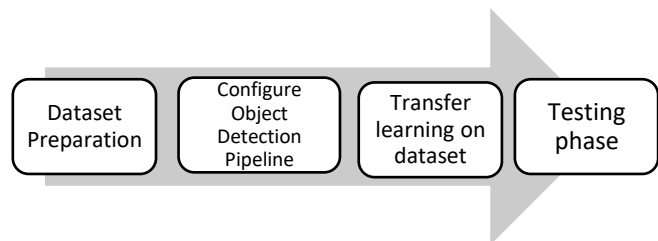


Fig. 1. Process Map for the Project. Pretrained Model and COCO dataset would be used along with TensorFlow and LabelImg to create object detection tool that detects vehicle.

than object classification since the former requires not only recognizing an object of a given class in an image but also

finding the coordinates of the object in the image and marking a bounding box around the recognized object. On top of that, it is imperative to prepare a properly annotated dataset to train the data for specific class of objects – in this case, vehicles.

Building a model from scratch requires great computational strength as well as takes time. Hence, available pre-trained models would be harnessed as the starting point for the project. The dataset is prepared and ensured proper annotations. Then, the TensorFlow API is configured with desired models and attributes. Once ready, the model is trained on the annotated dataset for detecting vehicles using popular deep learning models. The results are studied, analyzed and compared.

B. Approach

Two major development in the world of object detection using deep learning will the focus of this project. One is a Region Proposal Network (RPN) based Faster R-CNN and the other is a Single Shot Detector (SSD) [3][4]. These two approaches are selected for two main reasons. One, they are both very popular as well as efficient methods of object detections and hence, will provide results worth analyzing. Two, there are several pre-trained models of both the approaches that will help overcome the obstacle of properly training of the models due to computational limitations.

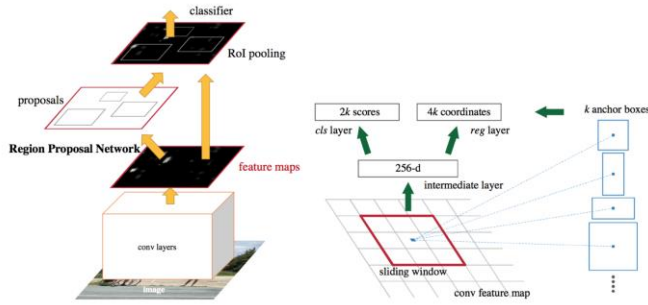


Fig. 1. Faster R-CNN. It utilizes Region Proposal Network (RPN) to extract a feature map from the input image after going through the classification and regression layers.

One of the most popular deep learning approaches for object detection is Convolutional Neural Networks (CNNs). RPN is the backbone of CNN (R-CNN), Fast R-CNN and Faster R-CNN are all built on top of CNN as the base with each of them utilizing better selective search methods [2]. RPN is a replacement for the selective search to generate region proposals. RPN makes use of anchors that are of specified ratios and located at fixed strides along the image. Each anchor is classified first into foreground or background. If the anchor (box) is classified as foreground, it's a go ahead to classify what lies inside the box. and refine the coordinates of the box [7]. Basically, a starter image is passed through a set of layers of convolutions with a specific weight matrix [11]. The aim is to extract features as a features map from the input image using the spatial arrangement of the images.

For Faster R-CNN, which is one of the selected models of this project, an input image is taken and passed on to a Convolutional Neural Network (ConvNet) which in turn outputs the feature map of the image. The feature maps are then applied

with region proposal network and generates a score that defines their objectness. A sliding window method is mostly used for RPN of each location [12]. Next, the results pass through an activation and Region of Interest (RoI) pooling layer which reduces the dimensions of the various proposals to the same size. Eventually, a fully connected network layer is used for these proposals to output the bounding boxes for the objects. The fully connected layer would usually consist of a softmax layer as well as a linear regression layer. The loss function for Faster R-CNN is [38],

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

| Symbol | Explanation |
|-----------|--|
| p_i | Predicted probability of anchor i being an object. |
| p_i^* | Ground truth label (binary) of whether anchor i is an object. |
| t_i | Predicted four parameterized coordinates. |
| t_i^* | Ground truth coordinates. |
| N_{cls} | Normalization term, set to be mini-batch size (~256) in the paper. |
| N_{box} | Normalization term, set to the number of anchor locations (~2400) in the paper. |
| λ | A balancing parameter, set to be ~10 in the paper (so that both \mathcal{L}_{cls} and \mathcal{L}_{box} terms are roughly equally weighted). |

The loss function is basically is the sum of the classification loss over 2 classes irrespective of the existence of an object or not and the regression loss of bounding boxes (this is only when an object exists) [5]. Thus, RPN checks beforehand if the image contains object and detection network gets the bounding boxes.

Another approach that will be studied and implemented in this paper is Single Shot Detection (SSD). SSD achieves similar or even better results than Faster R-CNN and demonstrates better time for doing so too [8][9]. Unlike, RPN based models which requires two shots, once to generate region proposals and then to generate detection of object for each proposal, SSD requires only a single shot to detect various objects in a given image [17].

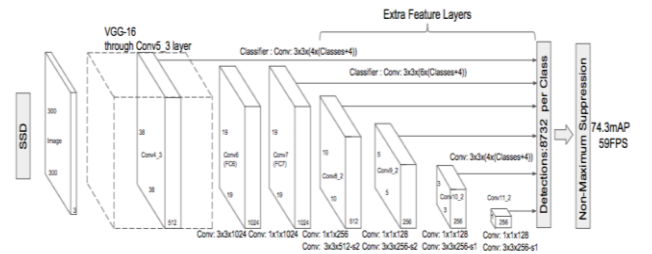


Fig. 2. Single Shot Multibox Detector. The architecture above is that of an SSD which utilizes multi-scale feature maps that retains only the best predictions.

In case of SSD, the input image is passed through several convolutions layers for feature extractions which results in a feature layer of size $m \times n$ with p number of channels. Many bounding boxes with various sizes and aspect ratios are then generated as well as class scores for each of these bounding

boxes is computed too. Different layers of the feature maps go through smaller convolution to give more and more accurate object detection. The below equation represents the loss function for an SSD,

$$L(x, c, l, g) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

The loss function is a sum of L_{conf} (confidence loss) and L_{loc} (localization loss) and N is the matched default boxes. The α is used to control the contribution of the localization loss.

C. Attributes

To see better results, it would be required to have to go through multiple iterations of the process of setting up the Object Detection API with a custom dataset [19]. For both the pre-trained models, that is, Faster R-CNN and Mobile SSD, the properties that are being set are tabulated in Table I.

TABLE I
ATTRIBUTES USED IN IMPLEMENTATION

| Neural Network → | Faster R-CNN | SSD |
|---------------------------|--------------|------|
| Batch-size | 1 | 24 |
| Speed (seconds per image) | 2.7 | 1.91 |
| Epochs to Converge | 320 | 60 |

R-CNN: Region based Convolved Neural Networks

SSD: Single Shot Detection

Batch size: determines the accuracy of the estimation of the error gradient.

Speed: Time taken to pass every image (seconds per image)

Epoch: One pass of the neural network over an entire dataset (forward and backward). Usually, divided into smaller batches due to computational limits.

Batch-size with manage the accuracy when estimating the error gradient while the neural network training is performed [22]. When we refer to a batch-size of X , it implies that X samples would be used in estimation of the error gradient from the train images dataset prior to the increase of the specific weights. Batch-size will usually exhibit contradictory accomplishments as compared to the speediness and stability of the transfer learning process [26]. For a pre-determined computational cost, it is claimed that a smaller batch size achieves a better training stability for faster R-CNN [27].

D. Tools to use

This project would be utilizing TensorFlow's object detection API which have various state-of-the-art pre-trained models already in place [13]. These data have been pre-trained on COCO data set, Open Images data set, Kitti data set, AVA v 2.1 data set and i-Naturalist Species Detection Dataset [14]. As the category of interest for this project, viz. vehicles, is already present in the dataset, further implementation will be done on top of this models.

After much analysis, the dataset to be used for this project is narrowed down to be COCO dataset as there are comparatively more variety of TensorFlow models trained in COCO dataset [36]. The models on which further implementation would be

done based on the approaches discussed in the previous sections are faster_rcnn_resnet101_coco and ssd_mobilenet_v2_coco.

This project will also utilize labelling tools to properly annotate the data for training. The tool used is LabelImg which is written in python and its GUI uses Qt. The annotations are converted as XML files in the format used by ImageNet.

Furthermore, a variety of software would be used to get the TensorFlow API up and running and to make a connection between the annotated data and the object detection framework.

III. PREPARATION

A. Setting up the system

This project is carried out on a Windows X 64-bit operating system. The installation of the latest version of Microsoft Visual Studio is the first step towards setting up the object detection environment. Windows doesn't support many Unix-based commands while at the same time, most of the installation procedures and instructions are unix-based. CygWin is installed to avoid wastage of time in converting unix commands into windows equivalent. Similarly, CMake is also installed to make way for a smooth installation of rest of the modules. Next, Github is installed as all the tools, starting from Labelling to TensorFlow requires Github for their setup. The language of choice for this project is Python and is used to set up the TensorFlow API as well. Hence, Python and pip are installed. VirtualEnv is installed to enable Virtual environment if necessary.

B. TensorFlow setup

TensorFlow developers has very detailed and well-explained instructions to setup the API in a local system [33][34]. The dependencies installed are Cython, ContextLib, Jupyter, Matplotlib, Pillow, lxml, and protobuf. Protobuf (Google's data interchange format) requires the maximum amount of time for the installation.

TensorFlow has several deep learning frameworks in their repository. However, only the necessary modules and models from the TensorFlow repository are required to be installed for this project. This project will require the Object Detection module and a few other dependencies such as the Slim module.

```
In [15]: # What model to download.
MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'

# Path to frozen detection graph. This is the actual model that is used for the
PATH_TO_FROZEN_GRAPH = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')

In [20]: # For the sake of simplicity we will use only 2 images:
image1.jpg
image2.jpg
# If you want to test the code with your images, just add path to the images to
PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(
# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8))
```

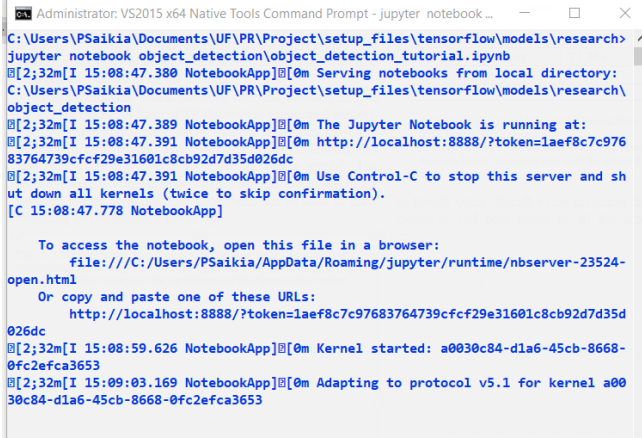
Fig. 3. Jupyter Notebook locations for modifying attributes and directories. MODEL_NAME can be edited to specify specific model. PATH_TO_LABELS and PATH_TO_IMAGES_DIR can be edited to point to custom train and test datasets.

TensorFlow also has a large set of models which are detailed in the TensorFlow detection Model Zoo. However, only the faster_rcnn_resnet101_coco and ssd_mobilenet_v2_coco would be necessary for the successful implementation of this project [15].

Finally, COCO Python API is installed that assists in loading, parsing, and visualizing the annotations in COCO. The exact format of the annotations is described on the COCO website [23].

C. Annotations

Annotations consists of laying down the coordinates of a vehicle and a corresponding label for that vehicle. Labellmg, a GUI tool will be used to prepare and annotate the dataset for



```
C:\Users\PSaikia\Documents\UF\PR\Project\setup_files\tensorflow\models\research>
jupyter notebook object_detection\object_detection_tutorial.ipynb
[2;32m[I 15:08:47.380 NotebookApp]@[0m Serving notebooks from local directory:
C:\Users\PSaikia\Documents\UF\PR\Project\setup_files\tensorflow\models\research\
object_detection
[2;32m[I 15:08:47.389 NotebookApp]@[0m The Jupyter Notebook is running at:
[2;32m[I 15:08:47.391 NotebookApp]@[0m http://localhost:8888/?token=1aef8c7c976
83764739cfcf29e31601c8cb92d7d35d026dc
[2;32m[I 15:08:47.391 NotebookApp]@[0m Use Control-C to stop this server and sh
ut down all kernels (twice to skip confirmation).
[C 15:08:47.778 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/PSaikia/AppData/Roaming/jupyter/runtime/nbserver-23524-
open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=1aef8c7c97683764739cfcf29e31601c8cb92d7d35d
026dc
[2;32m[I 15:08:59.626 NotebookApp]@[0m Kernel started: a0030c84-d1a6-45cb-8668-
0fc2efca3653
[2;32m[I 15:09:03.169 NotebookApp]@[0m Adapting to protocol v5.1 for kernel a00
30c84-d1a6-45cb-8668-0fc2efca3653
```

To access the notebook, open this file in a browser:
file:///C:/Users/PSaikia/AppData/Roaming/jupyter/runtime/nbserver-23524-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=1aef8c7c97683764739cfcf29e31601c8cb92d7d35d026dc

Fig. 4. Running object detection framework. Once TensorFlow and all its dependencies are completely installed and all the data is annotated properly, the Jupyter notebook object_detection_tutorial.ipynb can be run to view the results. This notebook has options to specify different model names, location of custom train and test data sets and other testing attributes.

this project [35]. Using Labellmg, bounding boxes around vehicles of interest can be drawn. Various such images were labelled to add to the efficiency of the vehicle detection. Labellmg easily generates annotation records in “XML” format. Then, Qdraw is used to convert the “XML” files to TensorFlow Records. The TensorFlow records can be used by the object detection API to train the pre-trained models over custom datasets of vehicles.

The test images were added to the test_images folder in the object detection module and the training datasets were added to the train_images folder. Once everything is set, the model name, the test and train dataset locations, as well as the various attributes are modified in the object detection jupyter notebook and all the cells are executed to test the object detection process.

IV. IMPLEMENTATION

A. Running the models

The implementation is started by training the faster R-CNN model with just 3 images containing vehicles. The training dataset size was gradually increased upto 34 images. Then, more images were labelled using Labellmg and fed into the neural network to obtain a test on a total dataset of 52 images. This number was further increased until the computation limit

of the system was reached.

The number of images in training dataset was increased to

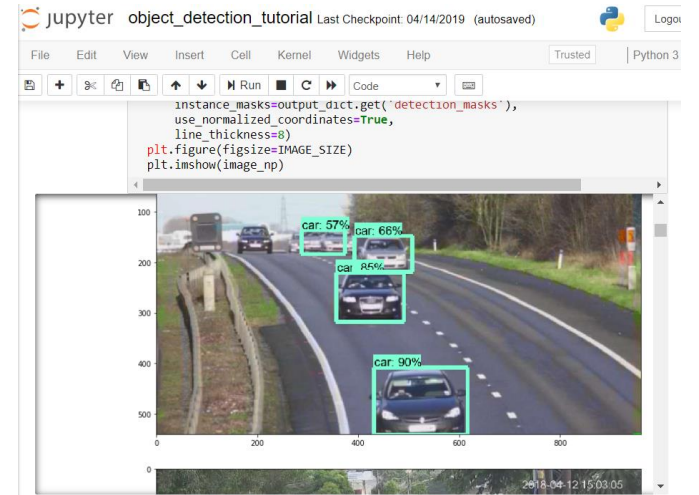


Fig. 5. Vehicle Detection. The final results of the vehicle detection after implementing the models can be viewed as output in the Jupyter notebook with bounding boxes around vehicles stating the confidence of the prediction

analyze if a larger well-labelled dataset improved the performance. If so, more and more label datasets can be used to increase the efficiency of the models [16].

After training the neural network over the labelled images (labels are in the form of TF Records as mentioned earlier), TensorFlow automatically saves models at various regularly intermittent checkpoints [25]. The checkpoint with the lowest loss was then used to generate a “frozen inference graph”. The frozen inference graph contains all the weights and biases of the network at that particular checkpoint and is used to draw bounding boxes around a test image to give the final output.

Tensorflow’s Tensorboard was used to keep live track of the loss curve to observe when convergence was achieved and stop the training process. The checkpoint file which is saved can then be converted into the frozen inference graph which is used on the test images to access performance. The faster R-CNN network was trained for 180 minutes for 320 epochs with a batch size of 1 till it converged whereas the SSD network was trained for 125 minutes with 60 epochs with a batch size of 24.

B. Attributes for measurement

It is necessary to analyze the results of the vehicle detection models over some of the most popular metrics used for measuring the accuracy of such models and algorithms.

The first attribute on which the results will be analyzed is Average Precision, also termed as AP in short [21]. It, as evident by its name, computes the accuracy of the predictions, that is the percentage of the output predictions that are correct. Mathematically, we have Precision as,

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

Another metric which will be used to measure the results on is Recall value. Recall value measures the number of all the positives and how many of all the actual results is being

detected in those positives. Recall value decreases when we move up the prediction accuracy.

Interestingly, it exhibits a zigzag pattern when it increases with false positives and decreases with true positives. Mathematically, we have Recall as,

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

V. RESULTS

A. Faster R-CNN

Faster R-CNN is much faster than its predecessors such as R-CNN or fast R-CNN. But not just that, faster R-CNN is also observed to have a high accuracy. Better performance is observed for Faster R-CNN model that is trained on 52 images than the one trained on 34 images. As more sets of images are tested, it is concluded that more the training dataset, the better the results. The 52 vs 34 images set will be used for reference in this paper. It is also observed that a smaller batch size indeed does provide better training stability for faster R-CNN for a given computational cost.

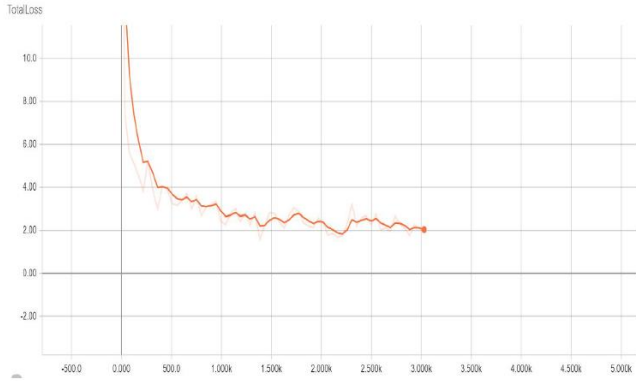


Fig. 6. Training Loss Curve using TensorBoard for Faster R-CNN (320 Epochs).

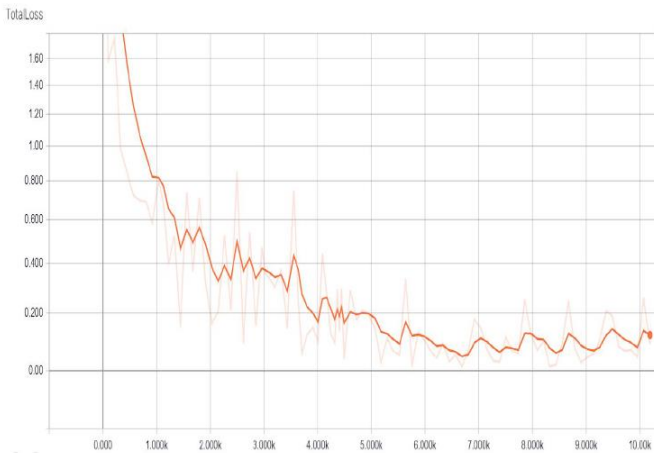


Fig. 7. Training Loss Curve using TensorBoard for Mobile SSD (60 Epochs).

B. Mobile SSD

Single Shot Detector achieves well when it comes to maintaining a balance amongst accuracy and speed. As mentioned earlier, SSD feeds the input image through a convolutional neural network just once and generates the feature map. For larger images of vehicles, such as in a traffic or in a highway, SSD performs better than in smaller images of vehicles such as satellite images or parking lots from a bird's view.

C. Comparison of both

The first important observation from the results is that SSD performs poorly on the test images in comparison to Faster R-CNN despite being trained on the same number of images. This is contradictory to what has been assumed at the start of the project. However, the advantage of SSD was that training time was much low and speed was higher in comparison to Faster R-CNN. This would allow for easy deployment of the model (for instance, with drones which requires quick responses).

Although SSD converged faster, the performance we obtain from the SSD is not very good in comparison to Faster RCNN.

Both the models performed poorly on datasets with car parking even when such images were a part of the training data. The faster R-CNN models performs well on both highways and residential areas (suburbs). However, the SSD model has a very low recall value (22.65% on test images) but higher precision in comparison to faster R-CNN. This means that the SSD model missed drawing boxes around several cars resulting a higher number of false negatives (because less boxes were drawn, lower number of false positives were observed and as a result, increasing the recall value). One interesting observation that was noticed in the 3rd image (a highway image) is that the Faster R-CNN that was trained on 34 images performed better on the (highway) image than the Faster R-CNN trained on 52 images.

TABLE II
COMPARISON OF RESULTS

| Neural Network → | Faster R-CNN | SSD |
|-------------------|--------------|--------|
| Average Precision | 71.3% | 83.33% |
| Average Recall | 46.39% | 22.65% |

R-CNN: Region based Convoluted Neural Networks

SSD: Single Shot Detection

Average precision: measures accuracy of predictions.

Average Recall: measures how well the model finds all the positives.

Precision and recall averages were calculated for both models as metrics for the object detector's performance. Precision tells us the ratio of TP and all the detections whereas recall gives us the no. of TPs from the actual no. of cars in the image. The values are illustrated in Table II. Increasing the no. of images did improve the performance of our Faster RCNN. This means that increasing the no. of examples to somewhere close to 1000 results in significant performance improvements.

VI. RELATED WORK

A. Power AI and Real-Time object detection

With the advancement of technology ever so fast, a lot of powerful cloud machine learning tools are making its way to enhance machine learning development [23]. One such tool that I came across while researching for this project is IBM Watson Machine Learning tool. This tool paves way to virtually utilize at an unlimited scale, the computational power and resource to optimize big algorithms. There is no need for local installations, cloning huge repos or even getting one's hands dirty annotating custom data sets, such as in our case [23] [31]. This tool provides all the resources from cloud annotations to training of large datasets. It accelerates the optimization of deep learning algorithms. The machine learning or deep learning workload can be as huge as one can imagine.



```

$ npm
Starting the development server...
Compiled successfully!

You can now view tensorflowjs-object-detection in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://192.168.0.22:3000/

Note that the development build is not optimized.
To create a production build, use yarn build.
  
```

Fig. 9. Real-time object detection using SSD model: TensorFlow.js used along with IBM Watson generated model.

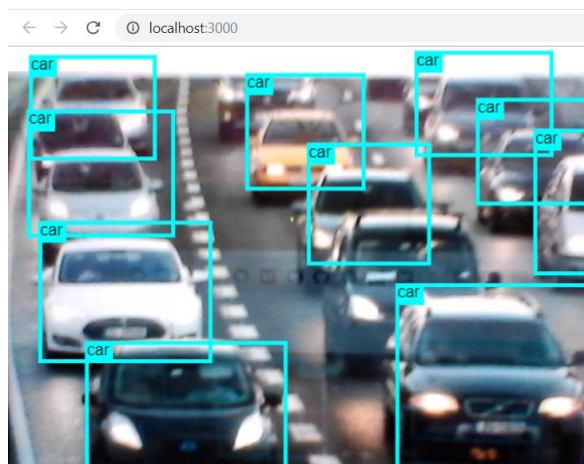


Fig. 10. Real-time object detection in webcam: A video from another device was used to feed the webcam to simulate a real-time moving vehicle detection environment.

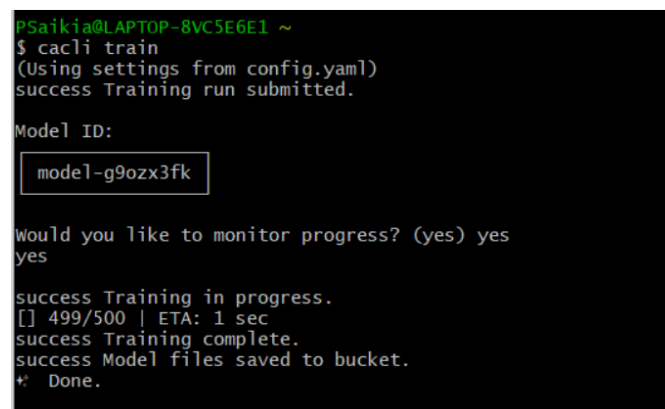
As an extension to this project, a basic vehicle detection tool was developed to test the already trained models in the Watson ML tool. It turned out that the process is extremely fast as well as very easy [37]. It starts with the creation of an object storage instance where the training and the test data is being stored. The cloud object storage provides credentials that can be accessed from a command line terminal to run various models. Then, a machine learning resource is created in the IBM Watson ML dashboard and another set of credentials are received. Many

buckets are created in the object storage resource to store different categories of training data. For this project, the annotation type is chosen as localization. For a classification problem, classification annotation type would have been used. Various images are uploaded in these buckets and are labelled as cars, bikes or trucks accordingly. For testing purposes, an SSD MobileNet model is downloaded and then trained via command line. As the training is completed, the model is download and is ready to be used in the app along with npm and tensorflow.js.

The saved time in the various processes can now be utilized to focus on more narrowed vehicle detection algorithms.

B. Real-time vehicle detection

Another important aspect in vehicle detection is doing it in real-time while keeping track of the surrounding environment too as a human eye would do. These kinds of advancements will lead to better traffic monitoring system as well as will pave way to exceptional self-driving vehicles [32]. Vehicle detection and surveillance puts forward a challenging task of segmentation of moving vehicles that too in multifaceted environment. The accuracy of detection in such cases would depend on the quality of the foreground masking used. “Real-time vehicle detection and tracking” puts forward a well-rounded approach to the problem [6]. The method that is proposed in the paper preserves a set of pixels in foreground that are marked as potential vehicles and rejects the remaining as noise. The experimental outcomes have recognized the better performance of the paper’s proposed algorithm.



```

PSaikia@LAPTOP-8VCSE6E1 ~
$ cacli train
(Using settings from config.yaml)
success Training run submitted.

Model ID:
model-g9ozx3fk

Would you like to monitor progress? (yes) yes
yes

success Training in progress.
[ ] 499/500 | ETA: 1 sec
success Training complete.
success Model files saved to bucket.
+ Done.
  
```

Fig. 8. IBM Watson Machine Learning Training for SSD model.

Again, to test the basic real-time vehicle detection using webcam from a laptop requires major changes on the interface side but rather minimal changes on the model configuration [24] [30]. For example, React js utilizing TensorFlow.js can be used with either with IBM Watson generated model or from a model generated from TensorFlow project to detect moving vehicles in videos [23].

VII. CONCLUSIONS

Faster R-CNN and SSD are both excellent choices for object detection at this time. However, Faster R-CNN establishes a slight advantage in accuracy over SSD. Single shot detectors, on the other hand are faster than faster R-CNN (ironically) and

are an excellent choice for real-time processing because of their speed. SSD demonstrates better detection of larger vehicles and gets worse as the vehicles in the images gets smaller. SSD also uses much less memory than faster R-CNN and hence, can be termed as lighter and faster.

Secondly, if the time distribution for various tasks is analyzed for this project, it is observed that annotation of custom dataset for vehicle detection takes the major chunk of time. While the pretrained models are exceptional, they are extremely generalized. It becomes necessary to get more specific to increase accuracy of a specific category of objects detected. For example, detection of only a certain color or car, or a certain model or size of car. For that, a large amount of time is spent on manual annotation of data. Hence, tools like IBM Watson Machine Learning and cloud annotation tool will come in very handy and reduce effort and time.

Finally, with the basic experiment using the IBM Watson machine learning tool, it can also be concluded that powerful AI tools can be used to harness the vast computational strength that cloud based tools brings with them. It will reduce unnecessary and tedious effort as well as save time. This would help in rapid development of efficient and fast vehicle detection algorithms and without redundant development.

All in all, this project provided exposure to the complete data-to-insight pipeline – from dataset preparation to data modeling. Also, the potential application of this project could be in traffic surveillance control, spying, disaster management, intelligent transportation systems, and designing logistics using drones.

ACKNOWLEDGMENT

I would like to thank Professor Dapeng Wu for his guidance and all the authors and researchers whose work enabled me to perform a successful implementation of the project.

REFERENCES

- [1] Prince Grover, "Evolution of Object Detection and Localization Algorithms," *Medium*, Feb 15, 2018, <https://towardsdatascience.com/evolution-of-object-detection-and-localization-algorithms-e241021d8bad>
- [2] Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks." In *Advances in neural information processing systems*, pp. 91-99. 2015.
- [3] Uijlings, J.R.R., van de Sande, K.E.A., Gevers, T. et al. : Selective Search for Object Recognition ", *Int J Comput Vis* (2013) 104: 154.
- [4] Girshick, Ross. "Fast R-CNN object detection with Caffe." *Microsoft Research* (2015).
- [5] Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580-587. 2014.
- [6] K. V. Arya, S. Tiwari and S. Behwal, "Real-time vehicle detection and tracking," 2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Chiang Mai, 2016, pp. 1-6.
- [7] Sik-Ho Tsang, "Review: Faster R-CNN (Object Detection)," *Medium*, Sep 14, 2018, <https://towardsdatascience.com/review-faster-r-cnn-object-detection-f5685cb30202>
- [8] Sik-Ho Tsang, "Review: SSD - Single Shot Detector (Object Detection)," *Medium*, Nov 3, 2018, <https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11>
- [9] Gao, Hao, "Understand Single Shot MultiBox Detector (SSD) and Implement It in Pytorch," *Medium*, Jun 6, 2018, <https://medium.com/@smallfishbigsea/understand-ssd-and-implement-your-own-caa3232cd6ad>
- [10] Hulstaert, Lars, "A Beginner's Guide to Object Detection", *DataCamp*, Apr 19, 2018, www.datacamp.com/community/tutorials/objectdetection-guide
- [11] Yang, Bo, et al. "Vehicle detection from low quality aerial LIDAR data", *IEEE Workshop on Applications of Computer Vision*, 2011.
- [12] Gao, Hao, "Faster R-CNN Explained", *Medium*, Sep 27, 2017, medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8
- [13] Yikang, Li, "Dataset gallery", *CVBoy*, Sept 20, 2017, cvboy.com/post/dataset/
- [14] Lin, Tsung-Yi, et al. "Microsoft coco: Common objects in context." *European conference on computer vision*. Springer, Cham, 2014.
- [15] Velde, Dion v., "How to train a Tensorflow face object detection model", *Medium*, Sep 26, 2017, towardsdatascience.com/how-to-train-a-tensorflow-face-object-detection-model-3599dcd0c26f
- [16] Chen, Xueyun, et al. "Vehicle detection in satellite images by hybrid deep convolutional neural networks." *IEEE Geoscience and remote sensing letters*, 2014.
- [17] Jonathan Hui, "SSD object detection: Single Shot MultiBox Detector for real-time processing", *Medium*, Mar 13, 2018, https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06
- [18] Eddie Forson, "Teaching Cars To See—Vehicle Detection Using Machine Learning And Computer Vision", *Medium*, Oct 25, 2017, <https://towardsdatascience.com/teaching-cars-to-see-vehicle-detection-using-machine-learning-and-computer-vision-54628888079a>
- [19] Vijendra Singh, "Custom Faster RCNN using Tensorflow Object Detection API", *Medium*, Oct 2, 2018, <https://medium.com/@vijendra1125/tensorflow-api-custom-object-detection-55444de6562d>
- [20] Karol Majek, "10 simple steps to Tensorflow Object Detection API", *Medium*, Jul 20, 2018, https://medium.com/@karol_majek/10-simple-steps-to-tensorflow-object-detection-api-aa2e9b96dc94
- [21] Jonathan Hui, "mAP (mean Average Precision) for Object Detection", *Medium*, Mar 6, 2018, https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173
- [22] Sagar Sharma, "Epoch vs Batch Size vs Iterations", *Medium*, Sep 23, 2017, <https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9>
- [23] Nick Bourdakos, "Tracking the Millennium Falcon with TensorFlow", *Medium*, Nov 2, 2017, <https://medium.freecodecamp.org/tracking-the-millennium-falcon-with-tensorflow-c8c86419225e>
- [24] Harrison, "Streaming Object Detection Video - Tensorflow Object Detection API Tutorial", *PythonProgramming.net*, Oct 20, 2018, <https://pythonprogramming.net/video-tensorflow-object-detection-api-tutorial/?completed=/introduction-use-tensorflow-object-detection-api-tutorial/>
- [25] Harrison, "Introduction and Use - Tensorflow Object Detection API Tutorial", *PythonProgramming.net*, Jul 3, 2018, <https://pythonprogramming.net/introduction-use-tensorflow-object-detection-api-tutorial/>
- [26] Jason Brownlee, "How to Control the Speed and Stability of Training Neural Networks With Gradient Descent Batch Size", *MachineLearningMastery.com*, Better Deep Learning, Jan 21, 2019, <https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/>
- [27] "Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD and YOLO)", *mc.ai*, Mar 28, 2018, <https://mc.ai/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo/>
- [28] Pulkit Sharma, "A Step-by-Step Introduction to the Basic Object Detection Algorithms (Part 1)", *AnalyticsVidhya.com*, Oct 11, 2018, <https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/>
- [29] Arthur Ouaknine, "Review of Deep Learning Algorithms for Object Detection," *Medium*, Feb 5, 2018, <https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852>
- [30] Piotr Skalski, "In-Browser object detection using YOLO and TensorFlow.js", *Medium*, Jun 15, 2018, <https://towardsdatascience.com/in-browser-object-detection-using-yolo-and-tensorflow-js-d2a2b7429f7c>
- [31] Léo Beaucourt, "Real-time and video processing object detection using Tensorflow, OpenCV and Docker.", *Medium*, Apr 12, 2018, <https://towardsdatascience.com/real-time-and-video-processing-object-detection-using-tensorflow-opencv-and-docker-2be1694726e5>

- [32] Harveen Singh, “Vehicle Detection and Tracking using Machine Learning and HOG.,” *Medium*, Mar 17, 2018, <https://towardsdatascience.com/vehicle-detection-and-tracking-using-machine-learning-and-hog-f4a8995fc30a>
- [33] TensorFlow, “Tensorflow Object Detection APP”, Git code (2018), github.com/tensorflow/models/blob/master/research/object_detection
- [34] TensorFlow, “Defining your own model”, Git code (2017), github.com/tensorflow/models/blob/master/research/object_detection/g3doc/defining_your_own_model.md
- [35] tzutalin, “LabelImg”, Git code (2018), github.com/tzutalin/labelImg
- [36] CocoDataSet, “COCO API - Dataset @ <http://cocodataset.org/>”, Git code (2018), github.com/cocodataset/cocoapi
- [37] Cloud-annotations, “Real-Time Custom Object Detection with TensorFlow.js”, Git code (2019), github.com/cloud-annotations/object-detection-react
- [38] Lilian Weng, “Object Detection for Dummies Part 3: R-CNN Family”, Github, Dec 31, 2017, <https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html>