Traffic Detection Using Computer Vision

Jayasooryan Thundikandi Maroli
School of Computing and Information Science,
Anglia Ruskin University,
Cambridge, United Kingdom
jt885@student@aru.ac.uk

***Abstract*- Emerging autonomous car industry enormously relied on computer vision to detect the traffic and try to improve the precision of detection continuously. This paper conducted a study on how to improve the object detection algorithm specifically for traffic.**

A pre-trained deep learning algorithm is used for conducting the experiments. YOLOv3 is an open-source pre-trained model for object detection. This algorithm is retrained using custom data to find the improvements in the accuracy. Data is collected from open image dataset repository. Model parameters are changed according to the new dataset and retrained. Apart from all other neural networks, YOLOv3 is preferred for a reason; YOLOv3 detect objects instantly without trading the accuracy also, retraining the algorithm is less complex.

The paper presents the implementation and retraining of YOLOv3 algorithm with custom traffic dataset using OpenCV computer vision library. It explains how the model is retrained, data is prepared for the algorithm, accuracy obtained after retraining.

Keywords – Artificial Intelligence, AI, Computer Vision, Traffic Detection, Neural Networks, YOLOV3, Pre-Trained Model

I. INTRODUCTION

Autonomous car industry has grown very fast in a short span of time and the reason behind the advancement is artificial intelligence. Autonomous cars use computer vision technology to detect road lanes, obstacle, traffic and the traffic lights and symbols; the fundamental aspect of an autonomous car is artificial intelligence system. Tesla motors is the leading autonomous car manufacturer; they have the superior computer vision technology to drive the car as its own. Build an object detection model that can instantly detect traffic with high accuracy and make decision is the key requirement of a traffic detection model. Building such model can be a time-consuming task also it requires high volume of computational resources. The task can be done with less computational resources and short span of time. This paper presents the building and implementation of such model to achieve fast and accurate traffic detection model.

Increasing demand and the use of autonomous car can be considered as a step to future transportation, but these technologies are at its budding stage, computer vision model that helps autonomous car to drive is bit far away from a perfect model. Errors are common in artificial intelligence model, but life-critical tasks cannot tolerate even small amount of error because it may lead to human life loss. As explained by (Kulkarni, Dhavalika and Bangar, 2018), due to various conditions it may be difficult to detect traffic lights, variations in weather and lighting condition are a challenge to get proper vision. In traffic detection also same challenges are applicable, for example: on a foggy day it would be hard to detect vehicles from far and delay in detection could lead to delay in decision making and it may cause accidents. Overcoming these type of challenge needs a robust deep neural network architecture that could detect instantly and accurately. The motivation of this project is to train a deep neural network that is fast and accurate which could reduce the incidents in road.

During the research phase of this project there were few questions raised on this topic:

- How to build a fast and accurate model?
- Why other models fail to detect traffic in various conditions?
- How to reduce the error in traffic detection model?

YOLOv3 is the answer to the first question, YOLOv3 is one of the state-of-the-art object detection algorithms that could detect multiple objects instantly without trading the accuracy. Training a deep neural network from scratch would not give accuracy as much as YOLOv3 can offer. YOLOv3 has an added advantage that it could detect multiple objects in an image at a time, it doesn't need to iterate through the image to find all the objects. The key reason of model failing to detect traffic is data, the model did not see enough variations in the scenarios; model is trained to find patterns, once the pattern is found the model works based on that, any input that does not covered in this pattern could be an error, to solve this enormous amount of data is required. Even though YOLOv3 detects 80 objects it must be trained particularly for traffic, using a vanilla YOLOv3 won't solve the problem; it may fail when it comes to different weather condition or lighting. So, the algorithm should be trained with different car images in various conditions to avoid error during the detection, this would reduce the error during detection.

The primary objective of this project is to build a traffic detection deep neural network model that detects traffic instantly and accurately. The pre-trained model should be retrained specifically on traffic dataset to increase the accuracy, through this we achieve a fast deep learning neural network that could detect traffic with negligible error. Implementing such model would be helpful for autonomous cars to reduce the error while driving. Currently all the market leading autonomous car manufacturer are continuously working to reduce the error in the computer vision model and increase the reliability. Building a fast and accurate deep neural network is the way to achieve this.

II. LITERATURE SURVEY

A literature review was conducted to identify different techniques and approaches for traffic detection using a neural network. Methods for traffic detection are classified as image processing-based, machine learning-based, or map-based techniques. Even though the image processing approach is straight and uncomplicated, it undergoes critical phases such as thresholding and filtering. Any miscalculation in these phases may result in ambiguous outcomes. To overcome this,

machine learning-based methods and algorithms are used to detect traffics accurately and instantly.

(Redmon and Farhadi, 2016) introduced YOLO9000, a real-time object detection system that can detect over 9000 object categories. In this paper, the author proposed a new method using the hierarchical view of object classification which allows the combination of distinct datasets together. The joint training algorithm which is proposed allows training the object detectors for both detection and classification of the data. With batch normalization on all convolution layers helps to regularize the model. By using the anchor boxes to predict the bounding boxes, YOLO can detect up to thousand boxes per image. WordTree is used to combine multiple datasets sensibly. This helps to combine data from various sources and the joint optimization technique makes it possible to train simultaneously on ImageNet and COCO. While most detection frameworks rely on VGG-16, the YOLO framework uses a custom network based on GoogleNet architecture which results in faster and accurate detection. Thus, using YOLO9000 provides us a real-time, faster, and accurate framework for detecting more than 9000 object categories by jointly optimizing detection and classification.

(Redmon and Farhadi, 2018) the author proposed, YOLOv3, a new hybrid approach between the network used in YOLOv2, Darknet-19 for performing feature extraction called Darknet-53, which is more powerful than Darknet-19. Here, a state-of-the-art, faster, and accurate detector is introduced. YOLOv3 predicts boxes at 3 different scales. The system extracts feature from those scales using a similar concept to the feature pyramid network. Several convolution layers are added from the base feature extractor where the last predicts 3-d tensor encoding bounding box, objectness, and class predictions. Here, a feature map from earlier in the network is merged with upsampled features using concatenation. This method helps to get more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature map. This helps us to identify and detect traffic instantly and accurately. By using YOLOv3 and YOLO9000 specifically on the traffic dataset, helps to achieve a fast deep neural network that could detect traffic with more accuracy.

In the work by (Kulkarni, Dhavalikar and Bangar, 2018) a deep neural network-based model for reliable detection and recognition of traffic lights using transfer learning is proposed. Transfer learning is a machine learning technique where there is an enhancement in learning a new task by channeling the knowledge through a related task that has formerly been learned. It is a method in deep learning that allows eliminating the extensive computational and time resources that are essential to developing neural network models by using pre-trained models. Thus, using the faster R-CNN-Inception-V2 model via transfer learning to traffic detection helps to improve the accuracy, making the system reliable for real-time applications.

III. Data

Since the project is using YOLOv3 algorithm to build the deep neural network the dataset must be formatted specifically for this algorithm. YOLOv3 requires a specific format which a dataset should contain image and annotation text file for each image. The annotation file corresponding to each image should have the same name as the image file, for example if the image name is car.jpg then the annotation file name should be car.txt. annotation file also created in a specific format; it should contain class of the object, center coordinates, width, height of the image. An example given below:

2 0.5210595 0.480625 0.957881 0.96125

Class of the object is represented in integers; it ranges from 0 to (number of class – 1). Center coordinates, width and height are represented in float; it ranges between 0.0 and 1.0. Preparing a dataset like this is a tedious task, we can use any of the following approaches to shorten the data preparation time. One we can use LabelImg tool to draw bounding boxes around the object to get the center coordinates, width and height or we can directly download bounding box images from open image dataset repository, but this repository gives us the bounding box dimensions of the image; we need to convert each file to the YOLOv3 format. LabelImg is an open-source image labeling tool that can load bulk images and draw bounding box and convert it to YOLOv3 format; Open Image Dataset (OIDv4) is an open-source data repository that contains 600 classes of images that has 1.9 million images. This project has used images from open image dataset. The downloaded images have been pre-processed and prepared for training. Downloading data requires OIDv4 command line tool which can be cloned from GitHub repository. A python script is used to request the download to the repository, required classes of object and number of images can be specified as command line argument. Once the request is successful each class will download one by one.

IV. Data Pre-Processing

The downloaded raw data folder contains two sub directories: one for images and one for csv files. Csv file folder contains two csv files which are class-descriptions-boxable.csv contains classes of images and train-annotations-bbox.csv contains ImageId, LabelName and annotations, it also contains other parameters which is not used in this project. This raw data must be pre-processed and prepared for training. As mentioned earlier bounding box dimensions must be converted to YOLOv3 format, which is X center, Y center, width, height. The bounding box variables are used in the following equation to get the desired values:

$$\text{X center} = (\text{X Max} + \text{X Min}) / 2$$

$$\text{Y center} = (\text{Y Max} + \text{Y Min}) / 2$$

$$\text{Width} = \text{X Max} - \text{X Min}$$

$$\text{Height} = \text{Y Max} - \text{Y Min}$$

String class name is converted to integers, the class name range is [0 – (number of classes -1)]. Each image is iterated through this operation and created text annotation file. YOLOv3 require train and test file path in a separate text file to train and test the algorithm. Images are looped to get the full file path and stored in a text file called train.txt and test.txt. Configuration file must be created to specify all the configuration settings for the current training. Configuration file contains number of classes, train.txt file full path, test.txt file full path, class names file, and weight directory path. These created files should be saved in conf directory of darknet framework.

V. Evaluation

Object detection algorithms are measured in a unit called mean Average Precision (mAP). Calculation of mAP requires evaluation of classification performance and evaluation of

localization. For Evaluation Intersection of Union (IoU) is used as measure. IoU evaluate the overlap between two boundaries; it defines how well the predicted object detection matches with the ground truth. To calculate mAP a threshold value must set; In this case value is 0.5, it can be indicated as mAP50 or mAP@50; the threshold value determines whether the object is valid or not.

If IoU >= 0.5 True positive

If IoU < 0.5 False Positive

If the image is not detected even after the ground truth is available, it is classified as False Negative.

True Negative is where object is not detected, since it is not useful, True Negative is ignored.

YOLOv3 has in-built function to calculate mAP. Evaluation can be done using the command map, configuration files and trained weight file path is passed as command line argument to perform the evaluation.

VI. Methodology

The entire project has been built upon a pre-trained deep learning neural network. This algorithm works bit different from other neural network training methods. YOLOv3 require specific dataset format and all the configuration settings, parameters are specified in text file instead of direct coding. Transfer learning methods in TensorFlow user can directly access the layers to retrain the model, YOLOv3 depends on the weights file and configuration settings to retrain the networks. Most of the technical parts are handled by algorithm itself.

A. YOLOv3

It is the heart and core of this project, the state of the art object detection algorithm is used to retrain the traffic dataset to improve the accuracy of the algorithm. YOLO stands for You Only Look Once and v3 is the version; unlike other object detection algorithm scan the image at once and detect multiple object at a time. The algorithm contains 106 convolution layers (Kathuria, 2018) to extract features and detect the objects. We can devide the entire algorithm into two major components; one feature extractor and other one is detector (Yanjia Li, 2019). When an image is inputted into the algorithm the feature extractor gets the features from the image then the features are fed into detector to find the objects in image. In this way algorithm can detect objects at one shot.

B. Installing Darknet Framework

Darknet is the framework for YOLOv3 algorithm; The files can be cloned from GitHub repository (Redmon and Bochkovskiy, 2021). Once cloned, Makefile information must be changed to match your training settings; GPU, OpenCV, CUDNN flag can be set to 1 to improve the training performance and a simple “make” command could install the framework. OpenCV, CUDA dependency files should be installed to use these features. And a Graphics processing is required for fast training.

C. Preparing Algorithm for Training

Training YOLOv3 require three files: the configuration file, training parameter file and the pre-trained weights file. Pre-trained weights are available on YOLO official website (Redmon, n.d.). configuration file must be created as part of the data pre-processing and the parameter file should be updated as per the project requirement. Configuration file contains number of classes, train dataset full path file, test dataset full path file, class names, and path to save trained weight. Parameter file contains all the parameter that defines each layer, learning rate, batch size, number of iterations and many more. For training and testing purpose two parameter files can be created, in training file the following parameters are updated:

Training Parameter:

batch=64
subdivisions=32
max_batches = 2000
steps=1600,1800

Layer Parameters:

filters=18
classes=1

batch size and subdivisions can be increased based on the training resource capacity, max_batches are calculated using a formula

max_batches = (number of classes) * 2000

steps are calculated based on 80% and 90% of the max_batch sizes. Filters are the last layer filter which must be calculated using the following formula:

filters = (classes + coordinates + 1) * masks

coordinates and masks are not changed; it is calculated as:

filters = (1 + 5) * 3 = 18

The same steps are followed in test file also but batch and subdivisions are 1.

D. Training

Training the algorithm is ran by a command using the darknet framework. All the prepared files are pasted in respective directories and the weight is loaded. Experiments are conducted using the help of graphics processing unit and CUDA support. Once the weight is loaded and the training process started each batch of images are loaded and retrained to get the new weights.

VII. Experiments

Since the algorithm uses enormous amount of computational resources which is not available to conduct enough experiments, limited number of experiments are conducted. Total of 3 experiments performed on different datasets. As a baseline model YOLOv2 algorithm, improvement from this algorithm is expected by training traffic specific dataset.

For the first experiment 5 classes are selected and 1000 image for each of the class. As per the YOLOv3 configuration it should be running 10000 iterations. At 768^{th} iteration computation resources ran out and the training process failed.

TABLE I.

Number of Iterations	*Time to Train*	*No. of Images*	*mAP@50*
768/10000	6 Hours	5000	0.25 %

Fig. 1. First Training Metrics

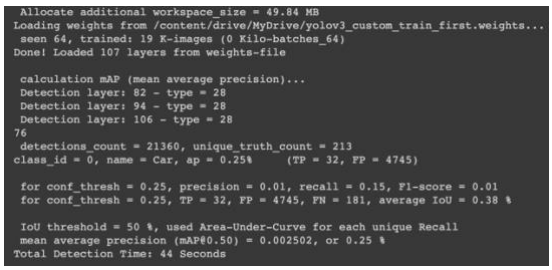

Fig. 2. First Model Metrics Screenshot

Second experiment had 100 images and single class; it ran for 2000 iterations and the performance was poor.

TABLE II.

Number of Iterations	*Time to Train*	*No. of Images*	*mAP@50*
2000/2000	4.5 Hours	100	0.19 %

Fig. 3. Second Training Metrics

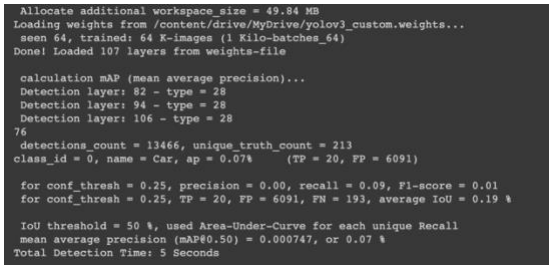

Fig. 4. Second Model Metrics Screenshot

The final experiment has better accuracy than the baseline model, even though it was trained with 500 images and 2000 iterations the model has adapted to the pattern and acquired better accuracy than the vanilla model. the baseline model has a mAP of 48.1% and our model increased the accuracy by 1% with this small dataset.

TABLE III.

Number of Iterations	*Time to Train*	*No. of Images*	*mAP@50*
2000/2000	9 Hours	500	49.11 %

Fig. 5. Final Training Metrics

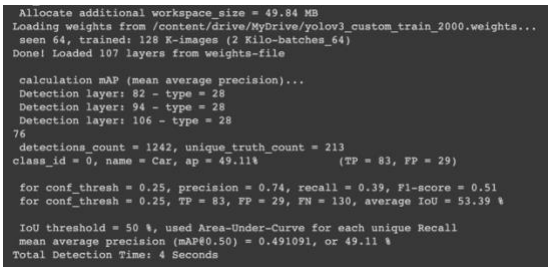

Fig. 6. Final Model Metrics Screenshot

VIII. CONCLUSION

From the experiments conducted, the accuracy of YOLOv3 algorithm has improved. Experiments indicates that accuracy of an algorithm can be improved by training the pre-trained model with a specific dataset. Even though the conducted experiments cannot showcase an outstanding increase is performance, the objective of the project has satisfied. And it is understood that the aim of this project is feasible. Based on the experiments conducted and from the results, the algorithm asks for more resources and data. The original plan seems to be working here but it needs more resources to build a perfect model. As per the literature review, the YOLOv3 algorithm is really a fast and accurate deep neural network model. It detects traffic instantly and accurately. In the future improvements more experiments must be conducted with massive computing resources and data, so that a perfect traffic detection algorithm can be built.

REFERENCES

[1] Kulkarni, R., Dhavalika, S. and Bangar, S., 2018. Traffic Light Detection and Recognition for Self Driving Cars Using Deep Learning. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8697819> [Accessed 12 April 2021].

[2] Kathuria, A., 2018. What's new in YOLO v3?. [online] Medium. Available at: <https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b> [Accessed 12 April 2021].

[3] Yanjia Li, E., 2019. Dive Really Deep into YOLO v3: A Beginner's Guide. [online] Medium. Available at: <https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e> [Accessed 12 April 2021].

[4] Redmon, J., n.d. YOLO: Real-Time Object Detection. [online] Pjreddie.com. Available at: <https://pjreddie.com/darknet/yolo/> [Accessed 12 April 2021].

[5] Redmon, J. and Bochkovskiy, A., 2021. pjreddie/darknet. [online] GitHub. Available at: <https://github.com/pjreddie/darknet> [Accessed 12 April 2021].

[6] Redmon, J. and Farhadi, A., 2016. YOLO9000: Better, Faster, Stronger. [online] arXiv.org. Available at: <https://arxiv.org/abs/1612.08242> [Accessed 12 April 2021].

[7] Redmon, J. and Farhadi, A., 2018. YOLOv3: An Incremental Improvement. [online] arXiv.org. Available at: <https://arxiv.org/abs/1804.02767> [Accessed 12 April 2021].