Edge-Based Street Object Detection

Sushma Nagaraj, Bhushan Muthiyan, Swetha Ravi, Virginia Menezes, Kalki Kapoor, Hyeran Jeon

Computer Engineering Department
College of Engineering,
San Jose State University
San Jose, CA, US

***Abstract*—Nowadays everything is becoming smart and intelligent with the help of Internet of Things (IoT) and artificial intelligence (AI). One of the promising applications of the integration of IoT and AI is smart city. The typical design pattern of smart cities is to install cameras and various sensors as many spots as possible and connect them to data center servers that can make smart decisions based on the inputs from the cameras and sensors. In such structure, network bandwidth may hinder the real-time processing because sensory data should be sent to the servers in the remote location. To solve this problem, recent a few studies demonstrated edge computing. In edge computing, IoT devices can handle basic recognition. Thus, only sophisticated inputs that are not handled by IoT devices are sent to remote servers. This paper further demonstrates that the edge computing can provide stand-alone processing power for handling one of the fundamental applications of smart city, which is street object detection. Correct detection of various street objects is the core function of traffic systems and public safety applications of smart cities. A convolutional neural network model is developed by training with traffic data captured at California and Nebraska. Our model detects 14 objects with 25% average accuracy on NVIDIA Jetson TX2.**

Keywords—deep learning, autonomous cars, object detection, Detect-Net, YOLO.

I. Introduction

This Machine learning has given machines the ability to process various tasks without human intervention such as playing chess, diagnosing cancer, recognizing various objects such as human face and many more. One of the major branches of machine learning is deep learning. With multiple layers of feature extraction, deep learning has enabled recognizing complicated patterns of target objects thereby solving various problems like human brain. One of the most trending applications that is accelerated by deep learning is traffic objects detection. Traffic objects detection is the core component of smart traffic control of smart cities and auto-pilot system of self-driving cars. These applications not only make the urban life convenient but also safe by reducing car accidents.

However, correct traffic object detection is not easy. The first hurdle is diverse shapes of different objects. On street, there are pedestrians, vehicles, bicycles, and traffic signals. Vehicles again have different shapes depending on their types such as sedan, truck, van, and suv. Traffic signals impose different meaning based on the lighted color such as red, yellow, and green. Thus, traditional machine learning that uses shallow network cannot recognize these objects altogether. With multiple layers of feature extraction, deep learning can be the solution of traffic object detection.

In this paper, we demonstrate a deep learning model that recognizes 14 objects that are captured by traffic cameras. Different approaches were being used to achieve this task. Approach one was working with Digits and used DetectNet model. We used pre-trained model which was best suited for detecting one class, but we found that the results were biased towards the classes which had more number of images. So, our second approach was to use Yolo model [2] with Darknet framework. Darknet is written in C and Cuda [8]. The state of the art says that it only looks once over the images to train the model.

Our model has small footprint to be deployed on a mobile GPU platform. Thanks to the huge advance in processor technology, mobile platforms are recently providing over one Tera-flops of performance [7]. These powerful compute engines enable edge computing. In edge computing, mobile devices serve a few core functions that used to be handled only by data center servers. Edge computing is beneficial for many real-time applications such as traffic object detection because it can server timely solution without needing to communicating with remote servers through slow network. We deployed the trained model on NVIDIA Jetson TX2 without any detection accuracy loss. The paper organizes as follows. Chapter II explains related work. Chapter III describes the design of our model. Chapter IV explains various approaches used for solving the problem of Traffic management. Chapter V is dedicated to results and conclusion. Chapter VI gives information about various references used for this task to accomplish.

II. RELATED WORK

The ideology of Deep learning and neural networks has been in existence since 1965 [4]. Since then researchers have been working on how to make it better and improve computing in machines. Deep learning is a technique where the machine is fed with an algorithm, typically called a neural network, and tons of data. Machine is expected to learn certain patterns in the data. It is then tested to identify the pattern in a completely new test data. User can choose between two types of learning: Supervised and Unsupervised. When the data is labelled and provided for deep learning, it is called supervised learning else unsupervised learning. Deep learning has been very successful in solving different image classification, object detection and many other problems. We are trying to solve an object detection in traffic as a problem.

Object detection is one of the most challenging problems in computer vision and is the first step in several computer vision applications. Two well-known models for object detection are

Detectnet [1] and YOLO [2]. The real scenarios of objects being occluded, partially visible in an image, object scale, resolution of an image, viewpoint of the object in an image or video makes the problem of object detection more difficult [1]. Object detection is being solved since many days. In earlier days, object detection was being solved using feature based techniques. Recent approaches including RCNN uses selective search which is exhaustive and computationally expensive [1].

III. PROJECT DESIGN

The goal of this project was to design an object detection model of high accuracy to detect traffic objects from images and videos obtained from surveillance cameras. Object detection using deep learning is training a model or a system to learn about the various features in the image using various algorithms. To obtain a model with high accuracy, we trained the model on two different kinds of models, namely DetectNet and YOLO model. To detect objects in any image dataset, the quality of dataset and selection of model and framework is most important.

A. Dataset

The dataset is provided by NVIDIA for the challenge. It consisted of images in three different resolutions 480p, 540p and 1080p. The images provided in raw data format had to be annotated first using the annotation tool. The objects in the images can be classified into 14 classes, namely, cars, SUV, traffic signals – red, green, yellow, bus, trucks – small, medium, large, pedestrian, group of people, motorcycle, bicycle and crossing. The dataset had 10,302 images in different resolutions and with different light conditions. After the images were labelled, they were divided into training and validation dataset and converted to formats compatible for the training models. For DetectNet model, images should be compatible to KITTI format and for YOLO model the images should be in darknet format.

B. Models

The two most popular networks, Detectnet and Yolo were taken as reference and were modified per the KITTI dataset format.

1. DetectNet:

DetectNet is a network architecture that simultaneously performs object classification and a regression to estimate object bounding boxes [1]. It is an extension to the popular GoogleNet network. We utilize this existing Detectnet model in our first approach to solve the problem of object detection in images related to traffic. We also have made some changes to accommodate multi class object detection. In the Detectnet architecture:

- Data layers ingest the training images and labels and a transformer layer applies online data augmentation.
- A fully-convolutional network (FCN) performs feature extraction and prediction of object classes and bounding boxes per grid square.
- Loss functions simultaneously measure the error in the two tasks of predicting the object coverage and object bounding box corners per grid square [3].

Unlike image classification problems, we have to provide the bounding box coordinate information with each image for object detection problems. We have used the data sets provided by NVIDIA for different resolutions 480, 540 and 1080 to train our neural networks.

After training detectnet for multi class object detection we could not achieve a good accuracy and the results were always biased to one class. Hence, we switched to another model name Yolo. As Detectnet runs on the deep learning framework named nvCaffe, Yolo runs on another deep learning framework Darknet.

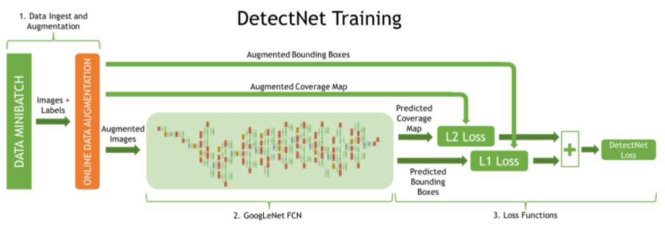

Fig. 1. DetectNet Training architecture [3]

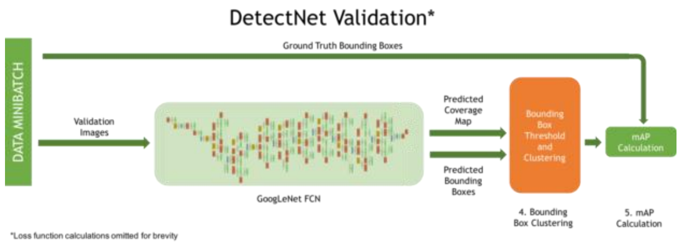

Fig. 2. DetectNet Validation architecture [3]

2. You Only Look Once (YOLO)

As opposed to other solutions proposed for object detection which repurposes the image classification classifiers, Yolo sees the object detection problem as a regression problem and uses regression techniques to solve the problem. In Yolo, a single neural network produces simultaneous predictions for multiple classes in one look. It encodes contextual information about the classes and their appearances and hence it is highly generalizable and applicable to new domains. [2] As shown in figure 3 Yolo network consists of:

- 24 convolutions layers
- 2 fully connected layers

1x1 convolution layers help in reducing the feature space from the upper layers.

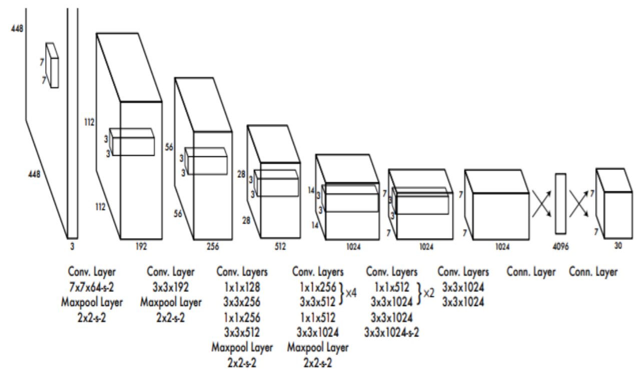
448
448
3
112
112
192
56
56
256
28
28
512
14
14
1024
7
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }x4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }x2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Fig. 3. Architecture of Yolo Neural net [2]

For training, the ImageNet classification pretrained model is used with resolution of 224x224 input images. For detection, the resolution is doubled. The first paper for Yolo [2], claims that their base network runs at 45 fps. It runs without batch processing. The experiments are performed on a Titan-X GPU. The fast version of the network runs at more than 150 fps. It seemed to be a very effective for our problem, if we could achieve the same performance. Hence, we decided to experiment with the NVIDIA provided dataset on Yolo.

IV. OPTIMIZATIONS

In this project, we have used two approaches for solving the issues. First, we had used DetectNet model with DIGITS framework and second was YOLO (You Look Only Once) model using DarkNet framework.

A. Approach 1: Using DetectNet

The Detect-Net model using caffe framework has been implemented using the steps shown in the following block diagram.

1) NVIDIA DIGITS

NVIDIA DIGITS is an interactive tool which makes it easy to use. It makes the task of managing data and training models user-friendly and easy. Monitoring performance and training models can be done on multiple GPU-system with ease. In our project, the aim was to detect objects in the traffic, which included different types of vehicles, along with people and traffic signal. Initially, the dataset was created in LMDB format using DIGITS. In order for the dataset to be compatible with DIGITS, it is converted to KITTI format using python script. The DetectNet model which was modified is then uploaded into DIGITS along with pre-trained GoogleNet model. Lastly, various parameters in DIGITS are modified to fine tune the model. DIGITS divides the image into various grids for DetecNet model. Size of each grid is slightly larger than the size of smallest object present in image for detection. Two types of information present in grid are the class of the object present and co-ordinates of boundary boxes. But if grid doesn't contain any class object, then it is considered as 'don't care'

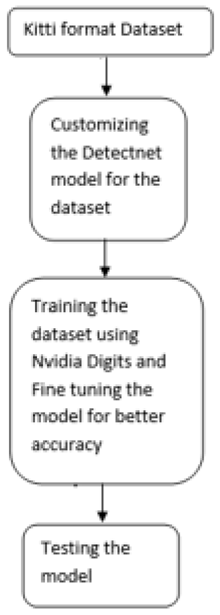
Kitti format Dataset
Customizing the Detectnet model for the dataset
Training the dataset using Nvidia Digits and Fine tuning the model for better accuracy
Testing the model

Fig. 4. Block Diagram showing steps of implementation for Detect-Net model

2) Modifying the DetectNet model for our dataset

DetectNet has been made to classify only one class. The dataset given to us contained images of size 1920x1080 pixels. But as we had to classify objects into 14 classes, following changes were made by us to make the model serve as per our requirements –

- The image-size-x and image- size-y has been changed to 1920 and 1080 respectively in the train-transfer and val-transfer method of data transformation layer
- A parameter associated with the dimension of the images has been modified as per the size of the images given in the dataset.
- As there are 14 classes, the output is changed accordingly.

At the end the cluster b-boxes and cluster-gt were added for all the 14 classes.

3) Fine Tuning our Model

The main advantage of DIGITS is that we can change the parameters for our model to improve its accuracy. Different parameters changes by us are learning rate, batch size, and solver type. Exponential decay is the parameter chosen as the learning policy and Adam as the type of solver, for our model. This helped us in increasing the efficiency of our model. Memory is another constraint. For this reason, batch size has been reduced accordingly to accommodate the batches that were used for training the model.

4) Results and Testing

After training the model for KITTI and Nvidia dataset, the performance of the model is as shown in Figure 5 below.

In Figure 4, the most important implication of accuracy of the model is mAP (mean average precision). The mAP is the difference between the predicted and actual output for the validation data set. Loss-bbox is the loss measures of the corner of the bounding box and loss-coverage is the presence of the object in the grid. The Precision is the ratio of true positives to sum of true positives and false positives. The recall is a ratio of true positives to sum of true positives and false negatives. The indication for the good performance of our model is if the mAP is close the either one. For our project, the model that has been developed has the value of mAP around 0.19 and the recall value around 0.22. The model is not satisfactory. An image is fed into the model as input for testing purpose. The result is as shown in figure 5.

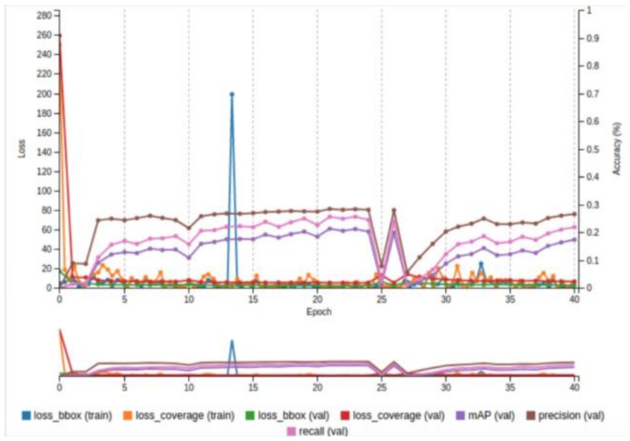

Fig. 5. Training curve for the Detect-Net model for Nvidia dataset

B. *Approach 2: YOLO*

1) *Converting dataset into DarkNet format*

In the Darknet format, there must be two text files. One is train.txt which stores the path for training images, and other is text.txt. For working with YOLO model with DarkNet framework, the given dataset had to be converted to the workable format for DarkNet

Also for this, the labels need to be saved as text files and their names should be exactly same as the name of its corresponding image in their original file extension. The labels must be in the following format – format: Class xmin ymin xmax ymax, where xmin, ymin, xmax, ymax are the coordinates of the bounding boxes. Confidence is the score of how much the model is believed to have an object in it. Thus, it will reflect a low confidence value if the bounding box does not have any object in it.

2) *Modifying the YOLO model for the VOC dataset*

We have used pre-trained YOLO weights as the pre-trained model for implementing the YOLO model. For customize the model as per the given dataset, the following changes have been made:

The classes (14 in our project) in the dataset is defined in VOC.name file under the /data directory.

In the VOC.data file, the number of classes were changed to 20 and the path of the text file is modified which contains paths to the train and the test images.

The number of batches and the subdivisions in the YOLO VOC.cfg represent the number of images processed at a time. One slight modification which is done is decreasing the subdivision, if memory is not sufficient and the process is getting killed.

Filter size is equal to the number of classes plus one multiplied by five. So for filter size has been set as (21) *5 = 125.

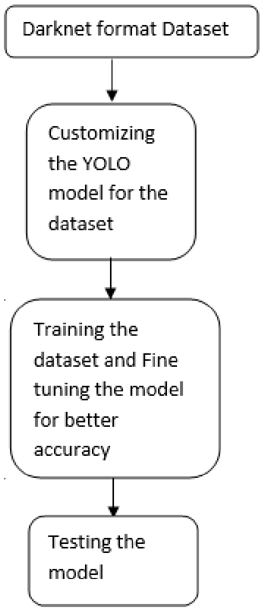
Darknet format Dataset
Customizing the YOLO model for the dataset
Training the dataset and Fine tuning the model for better accuracy
Testing the model

Fig. 6. Block Diagram showing steps of implementation for YOLO model.

3) *Fine Tuning the Model*

To fine-tune our YOLO model, we have tweaked the methods to achieve our goals. First, we stopped the training after certain iterations. Second, we used the previously trained weights as the pre-trained weights to start the training again. For the first ten thousand iterations, the learning rate of 0.001 has been used. Then for the next twenty thousand iterations, we reduced to 0.0001, and then further decreasing it to 2.5 e(-5). So, the learning rate was decreased exponentially.

4) *Results and testing*

The graph for the precision recall for Nvidia dataset is shown below:

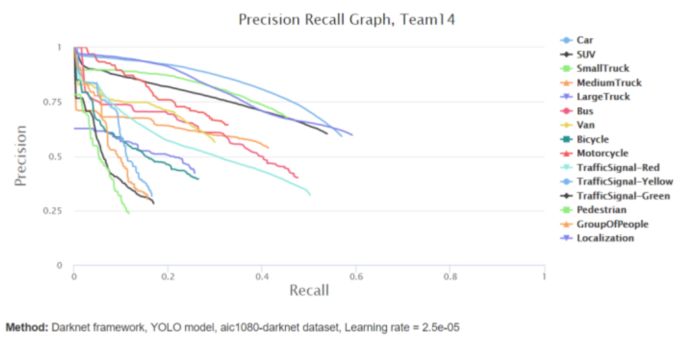

Fig. 7. Graph of Average precision and Recall for Nvidia dataset

V. DISCUSSION AND CONCLUSION

After training the model for YOLO as well as the DetectNet model, the mAP for the KITTI and the Nvidia dataset is shown the below table:

Model	Detectnet	YOLO
Map for Nvidia dataset	0.22	0.25

Table I. Mean Average Precision for each data set and models [6]

From the above table, it is clear that the performance for YOLO model was better than the DetectNet model. Yolo has the methodology of applying just one neural network across the entire image, which then divides the image into further regions and predicts the objects. The reason behind the better performance of YOLO is that it considers the context of the entire image while predicting the outcome. YOLO considers object detection as a regression problem where as DetectNet considers it as a classification problem. Also, one more reason for fast processing of YOLO is because it is a unified model. YOLO has an average of 4 frames per second. But the accuracy of the model can be increased further. The future scope of this project is to try another detection model like Faster R- CNN [5]. Our mAP (mean average precision) was 0.25 [6]. The accuracy can be increased by including pre-processing in data for data transformation. Another way of improving accuracy could be by some data augmentation process for the classes that have a low number of annotations or measure taken for treating outliers. Feature selections can also help in increasing the accuracy of the model.

VI. REFERENCES

[1] J. Barker and S. Prasanna, "Deep Learning for Object Detection with DIGITS," Aug 11, 2016. [Online]. Available: https://devblogs.nvidia.com/parallelforall/deep-learning-object-detection-digits/.

[2] Redmon, Joseph, et al., You only look once: Unified, real-time object detection, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016

[3] A. Tao, et al., "DetectNet: Deep Neural Network for Object Detection in DIGITS," Aug 11, 2016. [Online]. Available: https://devblogs.nvidia.com/parallelforall/detectnet-deep-neural-network-object-detection-digits/.

[4] F. Li, J. Johnson and S. Yeung (2017), CS231n: Convolution Neural Network for Visual Recognition, [online]. Available: http://cs231n.stanford.edu/.

[5] S. Ren et al., CS231n: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, in IEEE transactions on pattern analysis and machine intelligence. 2017 pp. 1137-1149

[6] Smart-city-conference.com, 'Challenge Results', 2017. [Online]. Available: http://smart-city-conference.com/AICityChallenge/results.html

[7] Nvidia.com, 'NVIDIA Jetson Modules and Developer Kits for Embedded Systems Development', 2017. [Online]. Available: http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html

[8] J. Redmon, 'Darknet: Open Source Neural Networks in C', 2013-2016. [Online]. Available: https://pjreddie.com/darknet/