Vehicle Detection And System Tracking Using Yolo V3 Model, A Computer Vision Technique

R.Anandhi¹, G. Sekar², C. Kalaivani³ and N. Jayalakshmi⁴

¹Assistant Professor, Department of MCA, Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, Tamil Nadu, India.

²Assistant Professor, PG and Research Department of Computer Science, Dr. Ambedkar Govt Arts College, Chennai, Tamil Nadu, India.

³,⁴MCA Students-Shift I (2018-2021batch), Department of MCA, Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, Tamil Nadu, India.

Abstract
Vehicle Detection and System Tracking is a real-time embedded system that recognises different types of vehicles automatically. This system is currently widely used in a variety of applications. The proposed method was created to recognise high-resolution digital photographs using the most up-to-date Computer Vision and Machine Learning approaches. A comparison of the different extant Computer Vision approaches utilised in YOLO v3 is made for this goal, as well as a detailed understanding of the operation and mode of use of the most often used Machine Learning algorithms, which are: Artificial Neural Networks, You Only Look Once (YOLO). Furthermore, a large vehicle picture dataset is required for the creation of an efficient, rapid, and trustworthy YOLO v3 model.

Keywords: YOLO V3, YOLO V2, Number Plate Detection, Darknet, Object Detection, Localization.

1. INTRODUCTION
The increased number of automobiles everywhere in today's scenario is obvious. During critical hours, it is extremely difficult to detect the numbers on the vehicle licence plate. As a result, this article proposes a Python method for detecting the vehicle number using the YOLO model. The YOLO (You Only Look Once) v3 model is a system capable of identifying vehicles without human intervention by using a high-speed image capturing technique with supporting illumination, detection of vehicles among the supplied images, and verification of the sequences as being those from a vehicle to convert image to xml [1], resulting in a set of metadata that identifies an image containing a vehicle and the associated
decoded images. Furthermore, for the development of an efficient, quick, and dependable YOLO, the phases adopted are as below:

- **Data Collection**: Data Collection is the initial process and much-needed process in the ML model dataset preparation. “Data is the key to unlock the future”. This critical and tough job is perfectly carried out by our great management team! Collected data will be usually in mp4 (video) format.
- **Data Analysis**: Now to our analytics team raw data and the list of features to be monitored are given. Then our analytics team will closely analyse the data and try to find the unusual pattern and finalize the features and classes name.
- **Data Extraction**: After finalizing the features and classes.txt next step is data extraction. In this step, convert the video data into a useful image format based on the analysed data.
- **Annotations**: After extracting the required data the next step is annotations. For annotations (labelling the image) we use 3rd part tools “LabelImg” or “Label Me”
- **Preparing datasets for training**: For training we needed to prepare a dataset (data.zip folder and upload it to g-drive for training). Google collaborator GPU is used here for faster training.

2. **MATERIALS AND METHODOLOGY**

Object identification is one of the fields that has profited greatly from recent advances in deep learning. One of the most important lessons learned from this experience is that the best approach to learn object identification algorithms is to build them from the ground up [2]. We use Darknet to create an object detector based on YOLO v3, one of the most efficient object identification algorithms available. Figure 1 depicts the steps involved in creating the YOLO v3 model:

- Creating the network architecture's tiers implementing the network's forward pass
- Non-maximum suppression and object score thres holding
- Creating the pipelines for the input and output

![Figure 1: Phases in Object Detection Procedure](http://www.webology.org)
Localization: Localization, like classification, determines the location of a single object inside an image. Many real-time problems can benefit from localization. Localization can be used to perform smart cropping (understanding where to crop images based on where the object is located) or even regular object extraction for further processing using various approaches. It can be used in conjunction with categorization to not only locate but also categorize an object into one of several possible categories [3].

3. YOLO v3 NETWORK ARCHITECTURE

Before a few years, YOLO 9000 was the quickest and one of the most accurate algorithms available. However, a few years later, it is no longer the most accurate algorithms, such as Retina Net and SSD, which have outperformed it in terms of accuracy. While the previous version of the game operated at 45 frames per second on a Titan X, the current version runs at around 30 frames per second [4]. This is due to the increasing complexity of the underlying Darknet infrastructure.

YOLO v2 employed a proprietary deep architecture called darknet-19, which consisted of a 19-layer network with 11 additional layers for object detection. YOLO v2’s 30-layer architecture made it difficult to detect small objects. The loss of fine-grained characteristics as the layers descended was attributed to this.

Detection at three Scales: Residual skip connections and up sampling are features of the newer design. The most notable feature of version 3 is that it detects at three distinct scales. The final output of a fully convolutional network, YOLO, is formed by applying a 1 x 1 kernel to a feature map [5]. As demonstrated in Fig 2, detection in YOLO v3 is done by applying 1 x 1 detection kernels to feature maps of three different sizes at three separate locations in the network. The detection kernel is 1 x 1 x (B x (5 + C) in shape. "5" is for the four bounding box characteristics and one object confidence, and B is the number of bounding boxes, a cell on the feature map.
Figure 2: Detection at Three Scales

The 82nd layer is the first to detect something. The image is down sampled by the network for the first 81 layers, with the 81st layer having a stride of 32. If we start with a 416 x 416 image, the resulting feature map will be 13 x 13. The 1 x 1 detection kernel is used to make one detection, yielding a detection feature map of 13 x 13 x 255. The feature map from layer 79 is then passed through a few convolutional layers before being up sampled by 2x to 26 x 26 dimensions. The feature map from layer 61 [7] is then depth concatenated with this one. The combined feature maps are put through their paces one more.

Better at detecting smaller objects: The issue of recognizing small objects, which is a common complaint with YOLO v2, is addressed via detections at different layers. The concatenation of up sampled layers with previous layers aids in the preservation of fine-grained features that aid in the detection of small objects. Large things are detected by the 13 x 13 layer, whereas tiny objects are detected by the 52 x 52 layer, and medium objects are detected by the 26 x 26 layer.

Choice of anchor boxes: YOLO v3 employs nine anchor boxes in total with three for each scale. If we are training YOLO on our data, we should construct nine anchors using K-Means Clustering and arrange them in descending order of dimension. Then assign the first scale’s three largest anchors, the second scale’s three and third scale’s last three [10].

More Bounding boxes per image: YOLO v3 predicts greater bounding boxes than YOLO v2 for the same input image size. For example, YOLO v2 forecasts 13 x 13 x 5 = 845 boxes at its natural resolution of 416 x 416. Five boxes were recognised using five anchors in each grid cell. YOLO v3, on the other hand, predicts boxes at three different scales. The number of expected boxes for the same image of 416 x 416 is 10,647. This means that YOLO v3 predicts 10 times as many boxes as YOLO v2. As a result, it is only slower than YOLO v2. Every grid can forecast three boxes using three anchors at each scale. Because there are three scales, there are a total of 9,3 anchor boxes used.

No more soft maxing the classes: Pay attention to the last three terms, despite the fact that the derivation appears to be frightening. The first penalizes the class prediction for bounding boxes that predict objects (the scores for these should ideally be zero), the second penalizes the objects score prediction for bounding boxes that predict objects, and the third penalizes the class prediction for the bounding box that predicts the objects. In YOLO v2, the last three terms are squared errors; however, in YOLO v3, they have been replaced by cross-entropy error terms. To put it another way, logistic regression is now used in YOLO v3 to forecast object confidence and class predictions. When we train, we give each ground truth box a bounding box.

4. DATA ANALYSIS AND INTERPRETATION
The COCO (Common Objects in Context) collection comprises demanding, high-quality visual datasets for computer vision, with the majority of them containing state-of-the-art
neural networks. MS COCO is a standard benchmark for measuring the performance of cutting-edge computer vision algorithms like YOLOv3 in the preceding example. At the COCO mAP (mean Average Precision) 50 test, YOLO v3 performs on par with other state-of-the-art detectors like Retina Net while being significantly faster. It also outperforms SSD and its derivatives [15].

In the ipynb (Interactive Python NoteBook) file once the darknet is installed then we need to wait for the same to be replicated in g-drive, usually it will take 2 to 8 min based on the size of the dataset we are going to train. Once the darknet folder is generated then we needed to go to “darknet->build->darknet->x64” and then delete the data folder. Come back to the darknet folder and find the unzipped data folder and move it to “darknet->build->darknet->x64 ” this path and then move the yolov3-tiny-obj.cfg which is inside the data folder into “darknet->build->darknet->x64” this path.

Above mentioned file movement to certain path inside darknet folder is mandatory as it is a part of a written documentary of darknet training. For more information regarding this you can refer to this github page “https://pjreddie.com/darknet/yolo/”. After file movement, we can run the remaining code in the ipynb file [16].

After running the training command “!./darknet detector train data/obj.data yolov3-tiny-obj.cfg -dont_show” We need to closely watch for any error message like the image can’t be loaded or the file is missing then we needed to check the image name mentioned in the error message and make the changes accordingly in the train.txt file or in the dataset.

If training is interrupted somewhere then we can continue the training based on the last weights stored inside the backup folder in the “darknet->build->darknet->x64->backup” mentioned path. Once the training is completed all the weights file are stored in “darknet->build->darknet->x64->backup” in this path. Now training is completed.

**Figure 3: Performance Comparison**

However, with a larger value of IoU (Intersection over Union) utilised to reject a detection, YOLO loses out on COCO benchmarks. In this case, 50 equals 0.5 IoU. The prediction is characterised as a mislocalisation and tagged as a false positive if the IoU between the forecast and the ground truth box is less than 0.5. In benchmarks with a higher number (say, COCO 75), the boxes must be properly aligned to avoid being rejected by the evaluation.
The following calculations show how to turn the network output into bounding box predictions.

**Object Score:** The chance that an object is included within a bounding box is represented by the object score. It should be nearly 1 for the red and neighbouring grids, and almost 0 for the grids in the corners, for example. Because the object score is to be read as a likelihood, it is also processed through a sigmoid.

**Class Confidences:** Class confidences indicate the likelihood that the discovered object belongs to a specific class (Dog, cat, banana, car etc). YOLO used to softmax the class scores before version 3. In v3, however, the design option has been eliminated, and authors have now chosen to use sigmoid. The reason for this is that while Softmaxing class scores, the classes are assumed to be mutually exclusive. To put it another way, if an item belongs to one class, it can't possibly belong to another. This is true for the COCO database, on which our detector will be based.

**Prediction on multiple scales:** YOLO v3 predicts on three separate scales. The detection layer is applied to feature maps of three different sizes, with strides of 32, 16, and 8 respectively. This means we make detections on scales 13 x 13, 26 x 26, and 52 x 52 with an input of 416 x 416. The network down samples the input image until it reaches the first detection layer, where feature maps from a layer with stride 32 are used to make a detection. Layers are also up sampled by a factor of two and concatenated with feature maps from prior layers that have the same feature map size. A new detection is made with stride 16 at layer. The same up sampling process is used again, and a final detection is conducted at stride 8 layer. Each cell predicts three bounding boxes using three anchors at each scale, for a total of nine anchors. (The anchors differ depending on the scale.) As a result, YOLO v3 improves its detection of small objects, which was a common criticism with previous versions of YOLO. Up sampling can aid in the network's learning of fine-grained features that are critical for detecting small objects. YOLO forecasts a size of 416 × 416 pixels for an image. YOLO predicts ((52 x 52) + (26 x 26) + 13 x 13) x 3 = 10647 bounding boxes for a 416 × 416 image [17].

**Object Confidence Thresholding:** First, we filter boxes depending on their object score. Boxes with scores below a certain level are usually ignored.

**NMS (Non-Maximum Suppression):** NMS aims to solve the problem of numerous image detections. For example, the red grid cell's three bounding boxes may all detect a box, or nearby cells may detect the same object.

5. **CONCLUSION**
This paper discusses about the vehicle number plate detection using YOLO v3 model provided the chip is embedded in the CCTV cameras installed in the traffic signals. This number plate detection will be useful in finding the vehicle which is the reason for accidents, theft vehicles etc. This model finds the vehicle number from the number plate in 2D view. This may be extended to have 3D modelling with post estimation method.

REFERENCES


