CS230

Video Vehicle Detection with YOLO

Jeffrey Gu
ICME
Stanford University
jeffgu@stanford.edu

Boning Zheng
Department of Statistics
Stanford University
b7zheng@stanford.edu

1 Introduction

Multi-object detection is the task of finding objects in an image or video frame, which consists of *object localization*, the problem of putting a bounding box around an object, and *object classification*, the problem of figuring out the class of the object. Object detection is important for understanding images and videos, and applications include image classification, face recognition, and human behavior analysis (1). Multi-object detection is also an important component of object tracking algorithms. General object tracking algorithms are important in autonomous driving, robotics, surveillance, and medical applications (2). The performance of state-of-the-art object detectors has improved in recent years thanks to the introduction of deep neural networks and novel object detection frameworks such as R-CNN, but these detectors are not specifically designed for video detection (3). Objects in videos display *temporal consistency*, which means that an object's location does not change much from frame to frame, and have *context*, which means that false positive detections are not likely to appear in many frames (3). Many techniques have been developed to use this information, such as tubelets (3) and sequential non-maximal suppression (4).

Most general-purpose object detection algorithms are very complex and thus typically do not run in real-time, which we define to be 30 frames per second (fps). For example, R-CNN, one of the high performance object detection architectures, takes 40 seconds per image at test time (5). One of the object detectors capable of running in real-time is YOLO (5). However, YOLO does not perform as well as the state-of-the-art object detectors (5). The aim of this project is to train YOLO for video vehicle detection and investigate techniques that may improve its performance and can be adapted to run in real-time. We trained two variations of the YOLO model for vehicle detection and apply a sequential non-max suppression to take advantage of temporal consistency in video data.

2 Related work

Most high performance object detection models are based on convolutional neural networks (CNNs). The three major models are Regions with Convolutional Neural Networks (R-CNN), YOLO, and Single-Shot multibox Detector (SSD) (3). R-CNN models and its successor solve the detection problem in stages, including generating bounding-box proposals, computing features for each region using a CNN, and then classifying each region. YOLO works by dividing an input image into a grid of evenly spaced cells and predicting bounding boxes and object scores for each grid cell (5). SSD uses set anchor boxes of different scales and aspect ratios to generate bounding boxes (3). Of the three models, YOLO and SSD are capable of running in real time but R-CNN is not.

Many methods have been proposed to take advantage of temporal consistency and other extra information available from videos. One idea, due to Galteri et al., (6) is to use detections and detector scores from the previous frame to re-rank region proposals for the current frame. Another idea, due to Kang et al., is to use tubelets, which are sequences of bounding box proposals, to exploit temporal consistency. Kang et al. propose a Tubelets with Convolutional Neural Networks (T-CNN) framework, which consists of of still-image object detection, multi-context suppression and motion-guided propagation, tubelet rescoring, and model combination (3). The second step leverages context to remove false postives and and consistency (via tubelets) to boost low scoring correct detections or to find non-detected objects. Tubelet rescoring is designed to increase the amount of time that tubelets can handle. The authors have also proposed another network to increase the speed of tubelet proposal generation (7). Sequential non-maximum suppression (seq-NMS) is another idea, which uses sequences of videos instead of single frames during non-maximum suppression (4), and is intended to boost weak detections using consistency. Our approach differs from their approach in that we will attempt to adapt Seq-NMS to work with the YOLO detection model (the authors of (4) use Zeiler-Fergus and VGG-16 as their detectors) and try to specialize to vehicle detections.

3 Dataset and Features

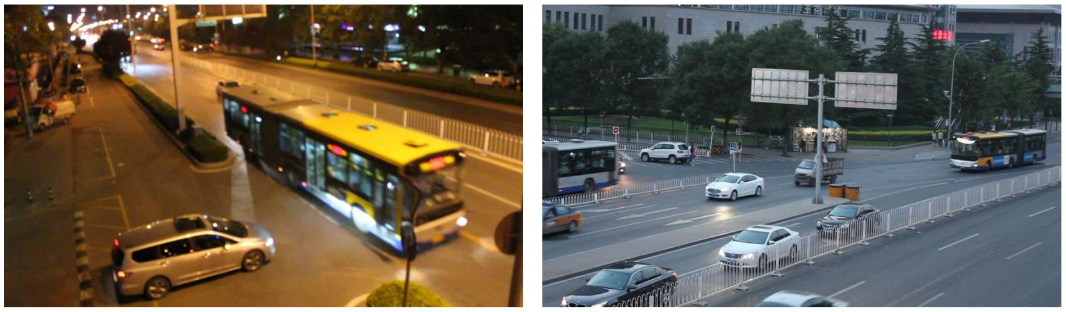


Figure 1: Example frames from videos in the UA-DETRAC dataset

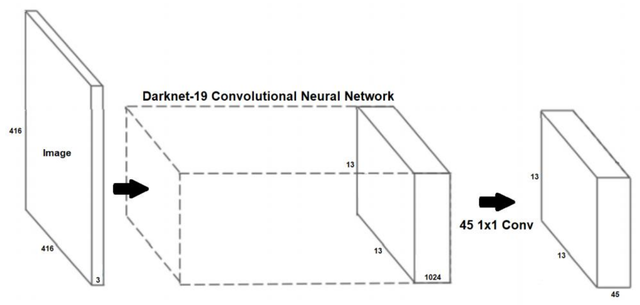
The University at Albany DETection and tRACking (UA-DETRAC) dataset is a multi-object detection and tracking dataset (8). The dataset consists of 100 videos of urban traffic, which were recorded at 25 fps at a resolution of 960×540. The videos are manually annotated, and in total there are 140,000 frames with 8250 vehicles and 1.21 million labeled bounding boxes. The test set consists of 40 video and is split by level of difficulty. The levels of difficulty are *easy* (with 10 such sequences), *medium* (20 sequences), and *hard* (10 sequences), where the difficulty is measured by the detection rate of the EdgeBox method (8). In addition to the bounding boxes, several other attributes are also annotated: vehicle category, weather, scale, occlusion ratio, and truncation ratio. Our pre-processing consisted of re-sizing all images were re-sized to a resolution of 416x416.

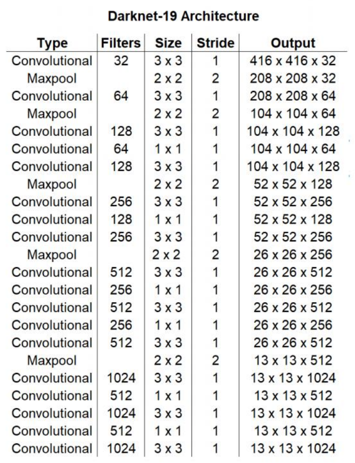
4 Methods

Our approach is as follows: first, we will create a baseline object detection model based on the YOLO architecture. We do this using the darkflow library, which is an open-source TensorFlow-based implementation of the YOLO model. Our dataset currently consists of 60 videos from the UA-DETRAC training set and their annotations. We split this set into 54 training videos and 6 testing videos. For the six testing videos, we chose six videos with a good mix of different times and weather conditions.

We first trained different variations of the YOLO object detection architectures to perform the object detection task, including YOLOv2 and Tiny-YOLO (9). Below is a summary of the YOLOv2 architecture. The architecture for Tiny-YOLO is similar, but only with 8 convolutional layers in the bulk of the network.

YOLO minimizes the following loss function:




Darknet-19 Architecture

Type	Filters	Size	Stride	Output
Convolutional	32	3 x 3	1	416 x 416 x 32
Maxpool		2 x 2	2	208 x 208 x 32
Convolutional	64	3 x 3	1	208 x 208 x 64
Maxpool		2 x 2	2	104 x 104 x 64
Convolutional	128	3 x 3	1	104 x 104 x 128
Convolutional	64	1 x 1	1	104 x 104 x 64
Convolutional	128	3 x 3	1	104 x 104 x 128
Maxpool		2 x 2	2	52 x 52 x 128
Convolutional	256	3 x 3	1	52 x 52 x 256
Convolutional	128	1 x 1	1	52 x 52 x 128
Convolutional	256	3 x 3	1	52 x 52 x 256
Maxpool		2 x 2	2	26 x 26 x 256
Convolutional	512	3 x 3	1	26 x 26 x 512
Convolutional	256	1 x 1	1	26 x 26 x 256
Convolutional	512	3 x 3	1	26 x 26 x 512
Convolutional	256	1 x 1	1	26 x 26 x 256
Convolutional	512	3 x 3	1	26 x 26 x 512
Maxpool		2 x 2	2	13 x 13 x 512
Convolutional	1024	3 x 3	1	13 x 13 x 1024
Convolutional	512	1 x 1	1	13 x 13 x 512
Convolutional	1024	3 x 3	1	13 x 13 x 1024
Convolutional	512	1 x 1	1	13 x 13 x 512
Convolutional	1024	3 x 3	1	13 x 13 x 1024

Figure 2: YOLO v2 Architecture

$$\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}$$

where (x_i, y_i) are the top left coordinates of the bounding box, h_i and w_i are the height and width of the bounding box, C_i is the box confidence score, and $p_i(c)$ is the class conditional probability. YOLO works by splitting an image into evenly spaced cells and having each cell predict a bounding box and an object score (the probability that there is an object in the bounding box) simultaneously. Before training, we initialized the weights for the YOLO models using pre-trained weights.

We also implement sequential non-maximum suppression (Seq-NMS) as a post-processing step for the YOLOv2 model. The algorithm for Seq-NMS has three parts: sequence selection, sequence re-scoring, and suppression (4). In analogy to single frame NMS, the first step consists of finding the maximum sequence of frames with respect to the sum of the object scores in each frame. To take advantage of temporal consistency, we also enforce the rule that adjacent frames have high overlap, which mathematically is the condition that the intersection over union (IoU) is greater than 0.5 (4).

$$i^* = \operatorname*{argmax}_{i_{t_s},\ldots,i_{t_e}} \sum_{t=t_s}^{t_e} s_t[i_t] \tag{1}$$

$$\text{s.t. } 0 \leq t_s \leq t_e \leq T \tag{2}$$

$$\text{s.t. IoU}(b_t[i_t], b_{t+1}[i_{t+1}]) > 0.5, \forall t \in [t_s, t_e) \tag{3}$$

We implement this using a dynamic programming algorithm, as suggested by the original authors of Seq-NMS (4). The second step is to re-score all the frames of the sequence with some function. (4) uses the average of the scores, the maximum score, and the best score out of single-frame NMS, average, and the maximum to re-score the sequence, but we will only use average re-scoring. Like single-frame NMS, the last step is to suppress frames that have high overlap with the frames in the sequence, in order to avoid repeat detections. The authors of the original paper (4) suppress frames with an IoU larger than 0.3, so that's the value we will also use.

5 Experiments/Results/Discussion

We trained two versions of the YOLO architecture for vehicle detection, YOLOv2 and Tiny-YOLO. To compare performance between models, we use mean average precision (mAP) as defined in (10), which is a weighted average of the average precision for each class.

To train the models, we used the Adam optimizer with learning rate 1e-5 decaying to 1e-6. All models were trained for 3 epochs, where the training/validation losses plateaued.

The overall performance of YOLOv2, the better performing of the two YOLO models, achieves a mAP score of 77%, compared to 62% for Tiny Yolo. For reference, the best performing YOLOv2 model benchmarked on the UA-DETRAC website achieves a mAP of 57.72%, but the training and test sets for the official benchmark are not the same as ours, so the numbers are not directly comparable.

Video	Time	mAP (%)
MVI_20051	Day	**81**
MVI_39861	Night	**61**
MVI_40181	Day	**88**
MVI_40732	Cloudy	**88**
MVI_41063	Day	**80**
MVI_63552	Day	**78**
	Overall:	**77**

Table 1: Summary of mAP scores for YOLOv2

We also tested Seq-NMS with YOLOv2 as the base detector on a small test set composed of a portion of MVI_20051, and found that Seq-NMS performs as well as traditional single-frame NMS but does not exceed its performance. A possible reason for Seq-NMS not exceeding the performance on traditional NMS is that Seq-NMS struggles with detector drift, where detections will drift between objects, which are usually close together and similar in appearance, leading to missed detections (4). In an urban traffic environment, there will be many frames with many cars close together, which can potentially cause many instances of detector drift, though our test set is too small to say anything conclusive. One potential way to fix this is to make the YOLO grid more fine, so it is harder for a detection to drift. The qualitative detections in Figure 3 are quite accurate but it seems to struggle

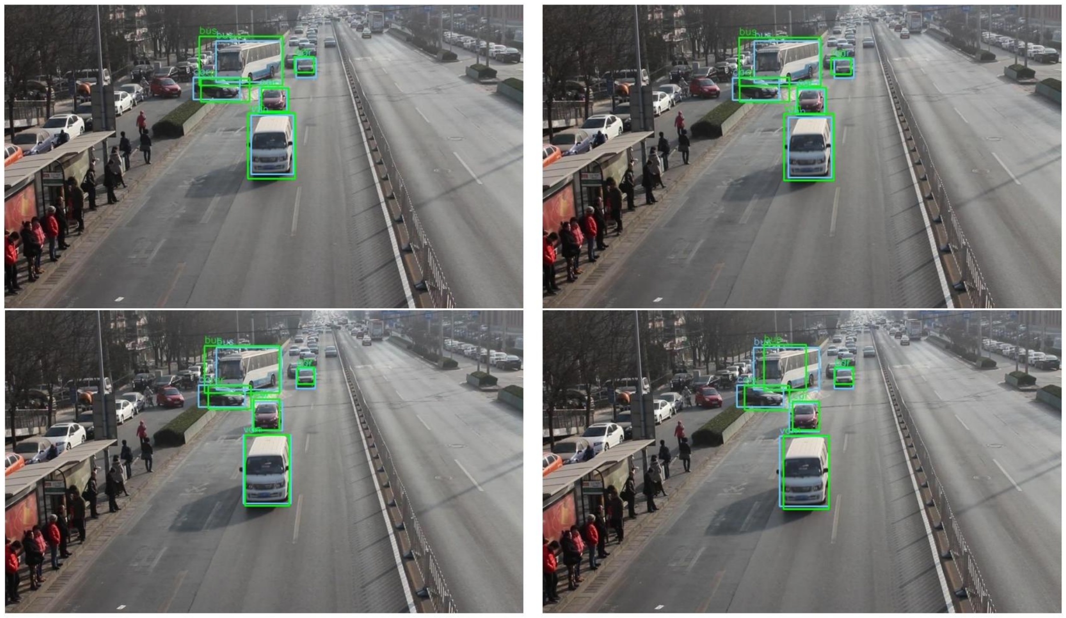


Figure 3: Detections and groundtruth in a single frame using Seq-NMS detection

with larger objects. If we examine the image of the bus, which looks like possibly the worst detection, it seems to include part of the car. This can lead spurious detections and false positives (4), and this may also be a common problem in urban traffic videos, as there will often be parts of other cars nearby, though as before our dataset is very small. The original authors of Seq-NMS have also identified this problem in their dataset, and this probably results from the fact that the objective function in Seq-NMS (Equations (1)-(3)) does not penalize adding more detections (4). A potential way to fix this problem is to penalize longer sequences.

6 Conclusion/Future Work

Over a very small test set of video vehicle detections, Seq-NMS performs as well as traditional single-frame NMS, and is a promising method to improve YOLO's video detections, both offline and possibly in real-time.

For future work, there are a few directions we'd like to explore. The first direction is that we would like to verify our results over a larger test set. Another direction is improving the speed of our code, as the current Seq-NMS implementation is in Python and very slow. Some ideas for speeding up the code include converting the code to Cython or another, faster language and paralellizing the code. Another direction is to try and adapt Seq-NMS for real-time detection by converting it to a streaming algorithm. We can also try combining YOLO with ideas from tubelets in order to improve video object detection. One such idea is to introduce context constraints to YOLO by suppressing false positive detections using nearby frames, as in (3), which can also potentially be adapted to run in real-time. Another is to use consistency during the running of YOLO itself, and not just in a post-processing step. For example, one idea is to use previous frames' detections to improve bounding box proposals, as in Galteri et al. (6)

7 Contributions

Jeffrey: Set up and managed AWS virtual machine instance, set up the darkflow and mAP Github repositories, helped train machine learning models, implemented, tested, and evaluated sequential non-maximal suppression, research, report writing

Boning: Data cleaning and preparation for model input, training YOLO on the UA-DETRAC dataset

Github: https://github.com/its-gucci

References

[1] S.-t. X. Zhong-Qiu Zhao, Peng Zheng and X. Wu, "Object Detection with Deep Learning: A review," *arXiv*, vol. abs/1807.00551, 2018.

[2] S. K. J. Mustansar Fiaz, Arif Mahmood, "Tracking Noisy Targets: A review of recent tracking approaches," *arXiv*, vol. abs/1802.02098v2, 2018.

[3] J. Y. X. Z. B. Y. T. X. C. Z. Z. W. R. W. X. W. W. O. Kai Kang, Hongsheng Li, "T-CNN: Tubelets with Convolutional Neural Networks for Object Detection from Videos," *arXiv*, vol. abs/1604.02532v4, 2017.

[4] T. L. P. P. R. M. B. H. S. J. L. S. Y. T. S. H. Wei Han, Pooya Khorrami, "Seq-NMS for Video Object Detection," *arXiv*, vol. abs/1602.08465v3, 2016.

[5] R. G. A. F. Joseph Redmon, Santosh Divvala, "You Only Look Once: Unified, real-time object detection," *arXiv*, vol. abs/1506.02640, 2015.

[6] M. B. A. D. B. Leonardo Galteri, Lorenzo Seidenari, "Spatio-Temporal Closed-Loop Object Detection," *IEEE Transactions on Image Processing*, vol. 26, 2017.

[7] T. X. W. O. J. Y. X. L. X. W. Kai Kang, Hongsheng Li, "Object Detection in Video with Tubelet Proposal Networks," *arXiv*, vol. abs/1702.06355, 2017.

[8] L. Wen, D. Du, Z. Cai, Z. Lei, M. Chang, H. Qi, J. Lim, M. Yang, and S. Lyu, “UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking,” *arXiv CoRR*, vol. abs/1511.04136, 2015.

[9] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” *arXiv*, vol. abs/1612.08242, 2016.

[10] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results.” http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.