AI City Challenge: Object Detection Using YOLO

Kevin Foley, James Frick, Vincenzo Moccia, Sanket Patel, Keri Wheatley

Introduction

The goal of the AI City Challenge is to make use of existing technology to improve the lives of the residents of the modern city. To ensure the greatest impact of these solutions, we will use existing infrastructure with novel techniques to produce meaningful insights.

By using data already being produced by the countless traffic cameras in modern cities, tasks from modeling traffic patterns to alerting emergency services of accidents in real-time can be implemented.

After initial models are created, researchers can then minimize the costs of implementation and realize even greater benefits by working to bring these solutions to the source of the data. These advances can then be retooled and implemented to provide solutions to improve traffic flows or even emergency departments.

Below are examples of use cases for the labeling and modelling of traffic camera data.

Scenario	Use Case
Identify traffic accident	Deploy real-time emergency services immediately after accident occurs
Identify assault or crime	Alert law enforcement and emergency services during or immediately after the incident
Identify wild or stray animal	Alert drivers via GPS technology; alert authorities and animal services
Record traffic patterns	Provide information for local government to make informed decisions on road infrastructure changes
Identify transportation modes	Research study to determine the availability of public and alternative transportation within a city

While the availability of raw data is exponentially increasing, the infrastructures to effectively utilize the data have not been built out. There is much potential for public safety improvements and resource allocation efficiencies using data that is already being recorded. The modelling and labelling of this data will incur upfront and subsequent maintenance costs that are outweighed by the research advancements and improvements to public safety.

Approach

Some of the most critical use cases above require real-time or close to real-time performance and a high degree of accuracy. This is something that computers have struggled with until recently. While R-CNN based solutions have achieved significant accuracy, they are still lacking in terms of detection speed.

For our solution, we chose YOLO (You Only Look Once), a real-time object detection system based on Darknet and developed by Joseph Redmon and Ali Farhadi[1].

Unlike other detection systems that repurpose classifiers to perform detection, YOLO applies a single neural network to the full image. The network divides the image into regions and predicts bounding boxes and probabilities for each region. As a result, we look at the whole image at once, which makes this approach very fast when compared to R-CNN based solutions.

A lighter version known as Tiny YOLO is also available. This version trades accuracy for speed and is suitable for inference on edge devices.

How YOLO works

YOLO is a convolutional neural network. However, unlike similar networks, there is no fully connected layer at the output.

Layer	Kernel	Stride
Input		
Convolution	3×3	1
MaxPooling	2×2	2
Convolution	3×3	1
MaxPooling	2×2	2
Convolution	3×3	1
MaxPooling	2×2	2
Convolution	3×3	1
MaxPooling	2×2	2
Convolution	3×3	1
MaxPooling	2×2	2
Convolution	3×3	1
MaxPooling	2×2	1
Convolution	3×3	1
Convolution	3×3	1
Convolution	1×1	1

YOLO divides up the image into a grid of 13 by 13 cells. Each of these cells is responsible for predicting 5 bounding boxes.

YOLO also generates a confidence score that describes how certain it is that the predicted bounding box actually encloses some object.

For each bounding box, the cell also predicts a class. This works just like a classifier: it gives a probability distribution over all the possible classes.

The confidence score for the bounding box and the class prediction are combined into one final score that tells us the probability that this bounding box contains a specific type of object. Most bounding boxes will have very low confidence scores, so we only keep the boxes whose final score is above a predefined threshold.

The very last convolutional layer has a 1×1 kernel and exists to reduce the data to the

[1] "YOLO: Real-Time Object Detection - Joseph Redmon." https://pjreddie.com/darknet/yolo/. Accessed 20 Aug. 2017.

shape 13×13×k, where 13×13 is the size of the grid that the image gets divided into, and k is the vector size of our predictions, including the bounding boxes, the confidence scores and the probability distribution over the different object classes.

Darknet

YOLO runs on top of Darknet, an open source neural network framework written in C and CUDA. Darknet supports both CPUs and GPUs and was created by Joseph Redmon[2].

Darknet can be used to implement popular image classification networks, such as AlexNet or VGG, as well as recurrent neural networks, mostly used for time series data and natural language processing.

Training YOLO/Darknet on the DGX-1

We ran our experiments on a VM running on top of a Nvidia DGX-1 deep learning system with dedicated access to one Nvidia Tesla P100 GPU.

For training, we used convolutional weights pre-trained on ImageNet. We used weights from the Darknet19 448x448 model which is based on the Darknet Reference network and Extraction model as well as other publications like Network In Network, Inception and Batch Normalization.

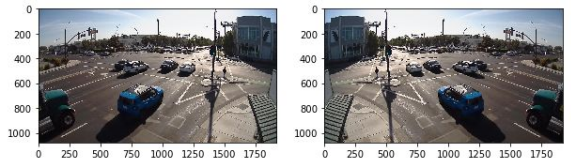
We experimented with mixing the aic480 and aic1080 datasets, but ultimately settled on training dedicated models for aic480 and aic 1080.

[2] "Darknet: Open Source Neural Networks in C - Joseph Redmon." https://pjreddie.com/darknet/. Accessed 20 Aug. 2017.

For each dataset we followed a similar approach to training. We split the train set into train and validation sets with a 90-10% split and used the validation set provided as the test set to evaluate our model. We used batch sizes of 64 and 8 subdivisions, primarily due to limitation of the GPU memory. We initially started training with learning rate of 0.001, momentum of 0.9 and decay of 0.0005 and later switched to a learning rate of 0.0001.

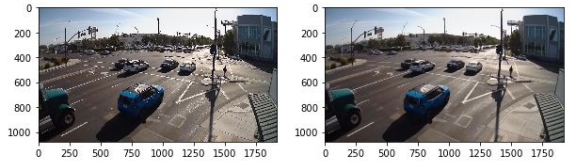
To help generalize our model better, we augmented the dataset and added them to our training set. Following are some examples of data augmentations used.

1. Horizontal Flip

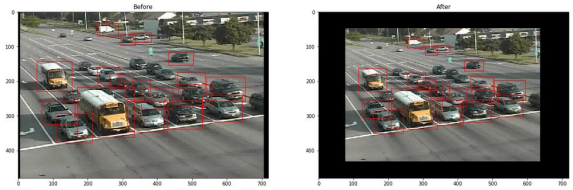


2. Vertical Flip

3. Gaussian Blur



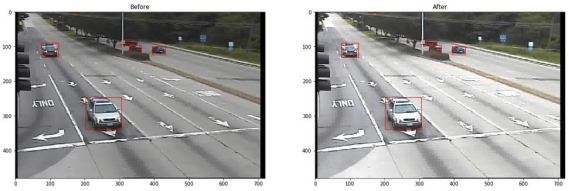
4. Scale



5. Shear

6. Coarse Dropout

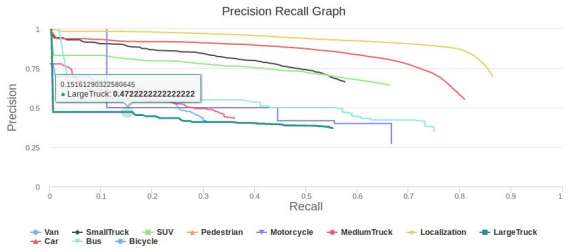
7. Pixel Multiply / Brightness



8. Elastic Transformation (sigma= 0.25)

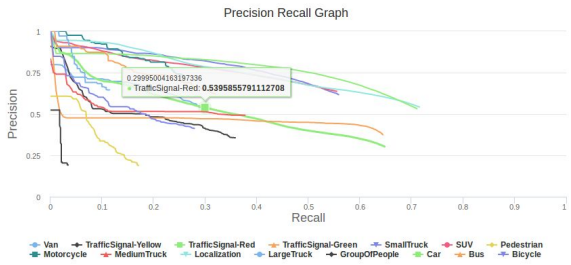
We ran the training for 45000 iterations. Following are the results for aic480 and aic 1080 test sets.

Aic 480 Test Set
mAP = 0.35



Class	AP	F1-score
Van	0.19	0.35
SmallTruck	0.48	0.62
SUV	0.51	0.65
Pedestrian	0	0
Motorcycle	0.37	0.39
MediumTruck	0.21	0.39
Localization	0.81	0.77
LargeTruck	0.24	0.44
Car	0.7	0.66
Bus	0.39	0.48
Bicycle	0	0

Aic 1080 Test Set
mAP = 0.27



Class	AP	F1-score
Van	0.2	0.38
TrafficSignal-Yellow	0.19	0.36
TrafficSignal-Red	0.35	0.41
TrafficSignal-Green	0.3	0.47
SmallTruck	0.45	0.59
SUV	0.44	0.59
Pedestrian	0.07	0.18
Motorcycle	0.23	0.38
MediumTruck	0.21	0.43
Localization	0.55	0.62
LargeTruck	0.09	0.19
GroupOfPeople	0.01	0.06
Car	0.55	0.61
Bus	0.25	0.42
Bicycle	0.16	0.33

Inference on the Jetson TX2

We ran the model on a Jetson TX2 device which has a Nvidia Pascal GPU with 256 CUDA cores and 8GB RAM. With the Yolo V2 model, we were able to get a frame rate of ~4fps on a 1920 x 1080 video and ~12fps on a 480 x 480 video. We also trained a Tiny YOLO model and were able to get frame rates of ~7fps and ~18fps on 1920 x 1080 and 480 x 480 videos respectively.

Conclusions

In this project, we used the Darknet framework and YOLO V2 architecture to detect and classify 9 different types of vehicles, pedestrians and traffic signals. Because one of our main focuses for the project was real time detection on the Jetson TX2, we picked YOLO as our preferred architecture over other state of the art architectures, like FRCNN. With data augmentation, we were able to achieve an mAP score of 0.35 and a frame rate of ~7fps on the Jetson TX2 device.

In the future, we plan to try other region-based proposal methods, like FRCNN.

References

[1] “Real-time object detection with YOLO”
http://machinethink.net/blog/object-detection-with-yolo/
[2] “YOLO: Real-Time Object Detection”
https://pjreddie.com/darknet/yolo/
[3] “You Only Look Once: Unified, Real-Time Object Detection”, by Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi
https://arxiv.org/abs/1506.02640v5
[4] “Darknet: Open Source Neural Networks in C”
https://pjreddie.com/darknet/