DIRECTED RESEARCH PROJECT
EE-590

Submitted By:
Mokshith Kumar Tumallapalli

Under the guidance of:
Prof. Bhaskar Krishnamachari
Mr. Kwame Wright

ABSTRACT

The goal of this project is to use video data to capture/model the traffic in a stretch of road using the YOLO + GPU pipeline. The main purpose of this project is to be able to find out what type of vehicles are moving in a stretch of road and to calculate the number of vehicles passing through a given region of interest. To capture and archive the real-time video, a camera can be used. The video file can then be used to process and find out of the required parameters. The video is fed to the application which then detects the vehicles in the video. The detection is achieved using the Darknet in the first part of the project and Haar Cascade classifier in the latter part.

The project can be implemented using the Darknet – an open source neural network framework or the OpenCV - an open source Computer Vision library. The proposed project has various applications in traffic engineering, security, and also in enhancing the inter-vehicular communications by extracting and making use of the data such as the distance between vehicles.

IMPLEMENTATION

The objective of the project to detect vehicles was achieved using the YOLO framework. YOLO - You Only Look Once is a state-of-the-art, real-time object detection system. Unlike the older detection systems, which repurpose the classifiers to perform detection and apply the model to an image at multiple locations and scales, YOLO use a totally different approach.

YOLO applies a single neural network to the full image. This network divides the image into regions and predicts bounding boxed and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. YOLO has several advantages over the classifier-based systems. It looks at the whole image at test time. It also makes the predictions with a single network evaluation.

A pre-trained model is used to detect the images. But the detection threshold can be changed from the default value which is set to 0.25 by passing the -thresh flag to the yolo command.

./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg -thresh 0.7

Fig 1. and Fig 3. shows the detected images using YOLO and their corresponding confidence values can be seen in Fig 2. and Fig 4.

Fig 1: Image Detected using YOLO

```
MOKSHITHs-MacBook-Pro:darknet mokshith$ ./darknet detect cfg/yolo.cfg yolo.weights data/dog.jpg
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   608 x 608 x   3   ->   608 x 608 x  32
    1 max          2 x 2 / 2   608 x 608 x  32   ->   304 x 304 x  32
    2 conv     64  3 x 3 / 1   304 x 304 x  32   ->   304 x 304 x  64
    3 max          2 x 2 / 2   304 x 304 x  64   ->   152 x 152 x  64
    4 conv    128  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x 128
    5 conv     64  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x  64
    6 conv    128  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x 128
    7 max          2 x 2 / 2   152 x 152 x 128   ->    76 x  76 x 128
    8 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256
    9 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128
   10 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256
   11 max          2 x 2 / 2    76 x  76 x 256   ->    38 x  38 x 256
   12 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512
   13 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256
   14 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512
   15 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256
   16 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512
   17 max          2 x 2 / 2    38 x  38 x 512   ->    19 x  19 x 512
   18 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024
   19 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512
   20 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024
   21 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512
   22 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024
   23 conv   1024  3 x 3 / 1    19 x  19 x1024   ->    19 x  19 x1024
   24 conv   1024  3 x 3 / 1    19 x  19 x1024   ->    19 x  19 x1024
   25 route  16
   26 conv     64  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x  64
   27 reorg              / 2    38 x  38 x  64   ->    19 x  19 x 256
   28 route  27 24
   29 conv   1024  3 x 3 / 1    19 x  19 x1280   ->    19 x  19 x1024
   30 conv    425  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 425
   31 detection
mask_scale: Using default '1.000000'
Loading weights from yolo.weights...Done!
data/dog.jpg: Predicted in 9.196120 seconds.
dog: 82%
car: 28%
truck: 64%
bicycle: 85%
MOKSHITHs-MacBook-Pro:darknet mokshith$
```

Fig 2: Confidence Values for the Detected Image

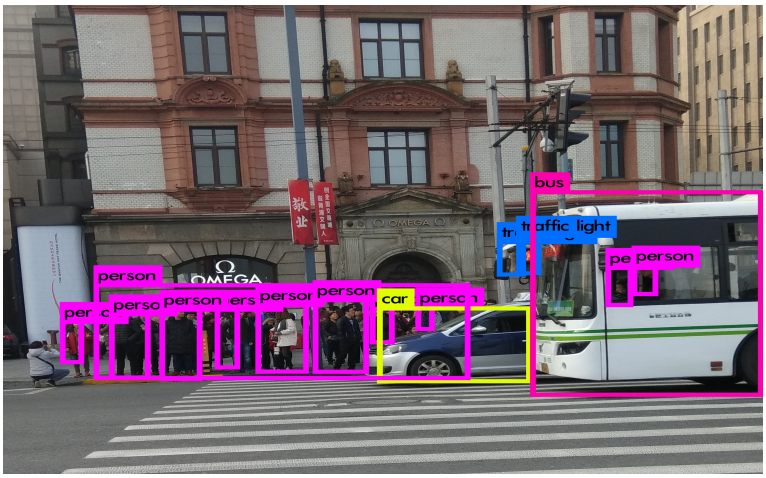

Fig 3: Image Detected using YOLO

```
   27 reorg              / 2    38 x  38 x  64   ->    19 x  19 x 256
   28 route  27 24
   29 conv   1024  3 x 3 / 1    19 x  19 x1280   ->    19 x  19 x1024
   30 conv    425  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 425
   31 detection
mask_scale: Using default '1.000000'
Loading weights from yolo.weights...Done!
data/img3.jpg: Predicted in 8.819539 seconds.
traffic light: 32%
traffic light: 35%
person: 37%
person: 40%
person: 25%
person: 31%
person: 28%
person: 40%
person: 56%
person: 51%
person: 30%
person: 41%
car: 78%
bus: 87%
person: 37%
MOKSHITHs-MacBook-Pro:darknet mokshith$
```

Fig 4: Confidence Values for the Detected Image

The second part of the project involved using the Haar Cascade Classifiers to detect the images in a video using OpenCV library.

Fig 5. shows the detected cars using the classifier. Pre-trained models are used for the detection. Several open source .xml files have been used and tested for this and finally the most suitable one is selected and is used for our project.

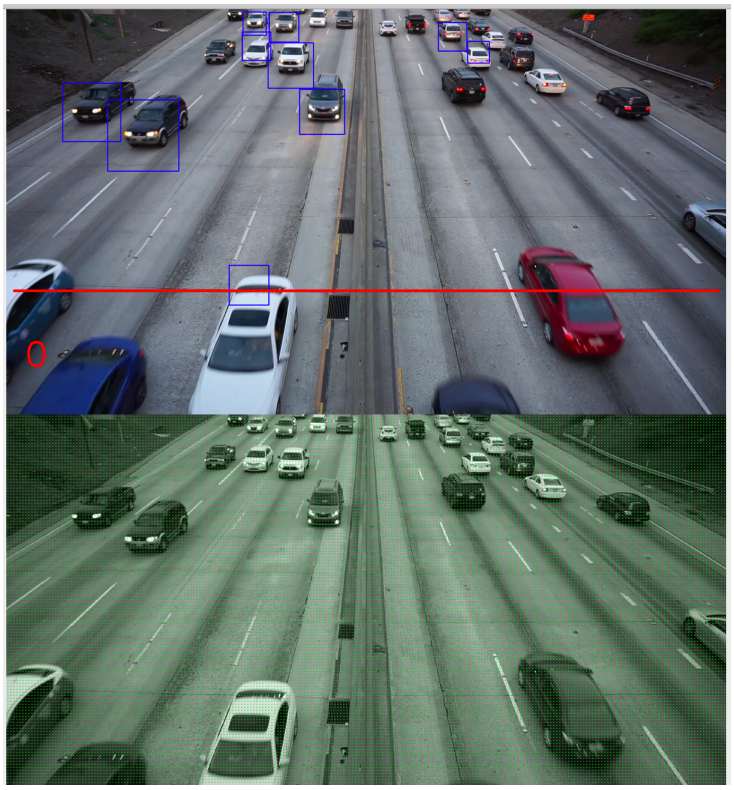

Fig 5: Vehicles Detected using Haar Cascade Classifier

In the figure above, a line has been drawn which is used to separate the region of interest. The region is below the line is considered as the region of interest in our scenario. Any vehicle moving past the line into the region of interest has to be noted.

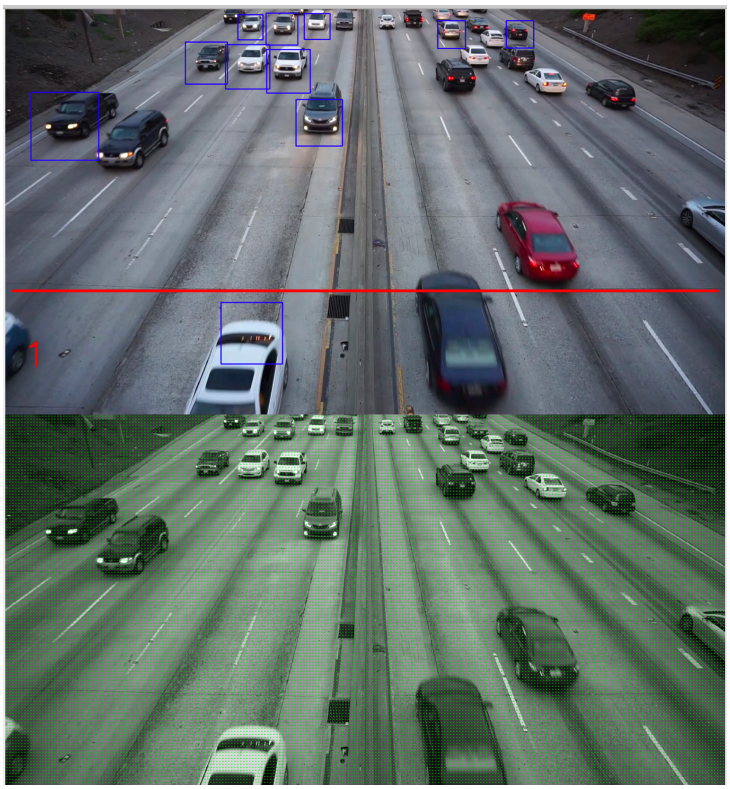

Fig 6: Vehicle Counter Implementation

From Fig 6. we observe that as the vehicle has passed across the region of interest, the count has been increased which is displayed on the image.

FUTURE WORK

The most immediate future work that could be done for the project is to refine the code and train a model with appropriate weights to suit our needs which produces less noise and jitter and detects the images perfectly.

After a reliable and efficient model has been built, it can be used to get more accurate results. Further debugging would have to be done to ensure that counter is increasing correctly as required and care must be taken such that the duplicate images from the successive frames are identified. Once that is completed, the obtained results can be used for several applications.

REFERENCES

1. https://pjreddie.com/darknet/yolo/

2. https://github.com/andrewssobral/vehicle_detection_haarcascades

3. Nilesh J. Uke, Ravindra C. Thool, “Moving Vehicle Detection for Measuring Traffic Count Using OpenCV,” in Journal of Automation and Control Engineering Vol. 1, No. 4, December 2013