A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers

Jonathan Pedoeem
Bachelor of Engineering in Electrical Engineering, The Cooper Union ’20

Abstract

A big focus of object detection has been to improve mean average precision leaving speed secondary. As a result, these algorithms need large GPUs to operate. YOLO-LITE addresses this problem. It runs 8.8x faster than Tiny-YOLOv2 at 21 FPS on a non-GPU laptop with an mAP of 33.6% making it an viable lightweight real-time algorithm.

Introduction

Object detection is the process of predicting bounding boxes and classifying objects. This field has been pushed to the bleeding edge with the rise of deep learning. The Imaginet classification competition has pushed the field of computer vision far, leading to impressive deep neural networks. These networks are large and require GPU computers.

Relevance

A quick lightweight object detection has many relevant applications. Augmented reality headsets need to be able to understand what its user is wearing in order to interact with her. Simpler objects, like thermostats, can use such an algorithm in order to assess if someone is in the room and then change the temperature.

Goal

To develop an object detection network running at 10 FPS on a non-GPU computer with 30% mAP on PASCAL VOC.

The Challenge

The challenge with this project was creating a small network that can have the desirable mAP and speed. The key was to find which elements of successful networks were necessary and to iterate through using combinations of them.

You Only Look Once

The You Only Look Once (YOLO) algorithm is a Convolutional Neural Network (CNN) developed by Joseph Redmon et al. YOLO is unique in that it was one of the first algorithms to do object detection by only processing image once instead of using a window or region selection protocol. This allows YOLO to be quicker than most other object detection algorithms [5]. YOLO’s speed and popularity made it the best choice as a starting point for YOLO-LITE.

Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classes</th>
<th>Train</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASCAL VOC</td>
<td>20</td>
<td>5,011</td>
<td>2,490</td>
</tr>
<tr>
<td>COCO [1]</td>
<td>80</td>
<td>40,775</td>
<td>4,995</td>
</tr>
</tbody>
</table>

Experimentation

Darknet, the framework created to develop YOLO was used to train and test the models. The training was done on a Alienware Aurora R7, with an intel i7 CPU, and a Nvidia 1070 GPU. FLOP counts were used as a predictor for FPS. There was no easy connection between number and size of layers and mAP. Reported FPS calculations are from a Dell XPS ’13 laptop.

Results

Seventeen trials were run before finalizing.

Comparison of datasets attempted while developing YOLO-LITE

<table>
<thead>
<tr>
<th>Dataset</th>
<th>mAP</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASCAL VOC</td>
<td>33.57%</td>
<td>21</td>
</tr>
<tr>
<td>COCO [1]</td>
<td>12.66%</td>
<td>21</td>
</tr>
</tbody>
</table>

State of Art on COCO

When it comes to real-time lightweight object detection algorithms this is how YOLO-LITE compares to state of the art on the COCO dataset:

<table>
<thead>
<tr>
<th>Model</th>
<th>mAP</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny-YOLOV2</td>
<td>23.7%</td>
<td>2.4</td>
</tr>
<tr>
<td>SSD MobileNet V1</td>
<td>21%</td>
<td>5.8</td>
</tr>
<tr>
<td>YOLO-LITE</td>
<td>12.26%</td>
<td>21</td>
</tr>
</tbody>
</table>

Conclusion

In conclusion, we were able to achieve our goal of a fast non-GPU algorithm. We also put up a live demo on the web which demonstrates the applicability and portability of YOLO-LITE. Our research has shown that shallow networks can run quickly on non-GPU computers with a tolerable mAP. We have found that batch normalization offers minor benefits in mAP while significantly slowing down shallow networks.

Future Work

The main focus of future work on YOLO-LITE is to improve mAP. Potential ways to improve mAP include

- Implementing depth-wise convolution [6]
- Pre-training on a classification dataset like Imagenet.
- Prune convolution filters [3]

References


Acknowledgements

This work was supported by the National Science Foundation DMS-1629328. Thanks to my partner Rachel Huang, PI Dr.Cuixian Chen, PI Dr.Yishi Wang, and GA Thai Thompson.

Contact Information

- Web: https://jped.github.io/
- Email: jonathanped@gmail.com
- Github: https://github.com/jped