YOLO-LITE

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 GPU computers 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.

Example Images Bounded by YOLO-LITE

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 Imagenet 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 asses 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.

Live Demo

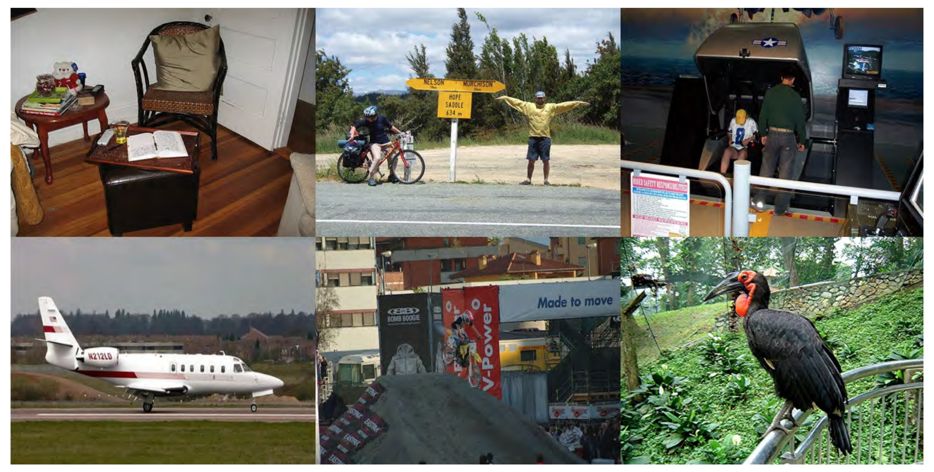
Demo: https://reu2018dl.github.io/

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

Dataset	Classes	Train	Val
PASCAL VOC [4]	20	5,011	2,490
COCO [1]	80	40,775	4,995



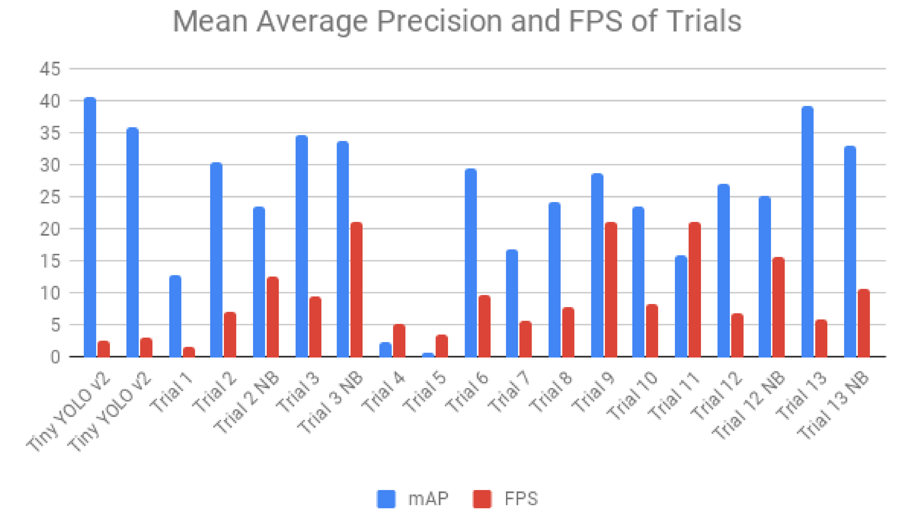
Examples of PASCAL VOC images

Experimentation

Darknet, the framework created to develop YOLO was used to train and test the models. The training was done on a Alienware Aura R7, with a 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 trials attempted while developing YOLO-LITE

Top Results

Dataset	mAP	FPS
PASCAL VOC	33.57%	21
COCO	12.26%	21

Results from trial 3 No Batch-Normalization (NB)

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:

Model	mAP	FPS
Tiny-YOLOV2	**23.7%**	2.4
SSD Mobilenet V1	21%	5.8
YOLO-LITE	12.26%	**21**

Comparison of SotA on COCO dataset

YOLO-LITE is **3.6 times** faster than SSD and **8.8 times** faster than Tiny-YOLOV2.

Discussion

Trial 3 NB exceeded both the mAP and FPS set out in the goal. Trial 3 NB implements three main changes over Tiny-yolov2:

1. Input images are 224x224 instead of 416x416.
2. One less layer and layers with smaller filters
3. No batch normalization. We found that batch normalization slows down forward pass of the network by about a half.

Batch normalization was introduced by Ioeffe et al. It is the idea of transforming the input space to have mean zero and standard deviation of one. They discussed that it can significantly increase the training time in deep neural networks while also offering minor improvements in accuracy [2].

Trial 3 No Batch Normalization

Trial 3 NB Architecture

1. 224x224 image into a 3x3x16 stride 1 conv layer w/leaky ReLU
2. 2x2 max pool with stride 2
3. 3x3x32 stride 1 conv layer w/leaky ReLU
4. 2x2 max pool with stride 2
5. 3x3x64 stride 1 conv layer w/leaky ReLU
6. 2x2 max pool with stride 2
7. 3x3x128 stride 1 conv layer w/leaky ReLU
8. 2x2 max pool with stride 2
9. 3x3x128 stride 1 conv layer w/leaky ReLU
10. 2x2 max pool with stride 2
11. 1x1x256 stride 1 conv layer w/leaky ReLU
12. 1x1x125 stride 1 conv layer w/ReLU
13. YOLO region layer

Conclusions

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

[1] COCO.
Coco - common objects in context.
http://cocodataset.org/, Last accessed on 2018-07-18.

[2] Sergey Ioffe and Christian Szegedy.
Batch normalization: Accelerating deep network training by reducing internal covariate shift.
arXiv preprint arXiv:1502.03167, 2015.

[3] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf.
Pruning filters for efficient convnets.
arXiv preprint arXiv:1608.08710, 2016.

[4] PASCAL.
The pascal visual object classes homepage.
http://host.robots.ox.ac.uk/pascal/VOC/index.html, Last accessed on 2018-07-18.

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi.
You only look once: Unified, real-time object detection.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[6] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun.
Shufflenet: An extremely efficient convolutional neural network for mobile devices.
arXiv preprint arXiv:1707.01083, 2017.

Acknowledgements

This work was supported by the National Science Foundation DMS-1659288. 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