YOLO “Real-Time” Object Detection on Low-Cost Hardware

Xiaolan Cai, Warren Ferrell, Chu-Sheng Ku, Soham Shah, and Ryan Skinner

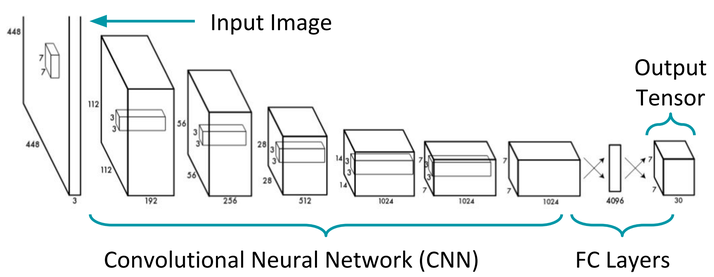
CSCI 5622 Machine Learning, Prof. Chris Ketelsen

Introduction

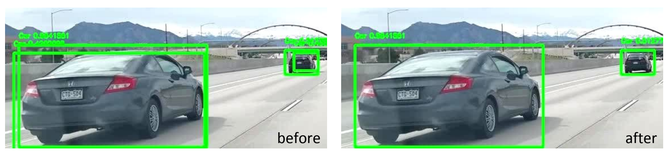
As autonomous vehicles become increasingly prevalent, it is critical that they operate effectively and safely. One major challenge to this is the high computational cost of accurate object detection systems. In 2015, the YOLO algorithm was the first to achieve real-time, high-accuracy object detection, running at ~100 fps on a Titan X GPU. This project seeks to reduce the cost of YOLO and optimize it for the performance-constrained environment of a Raspberry Pi.

YOLO (You Only Look Once)

- Neural network with CNN head, FC layers, and output tensor



- Output tensor contains object confidences and bounding boxes... for each anchor box shape, for each partition in the image
- Post-processing: non-max suppression (NMS) algorithm
  - If boxes overlap enough, keep only most confident one

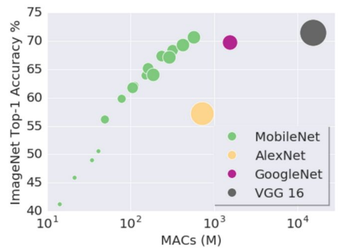


Training Data: KITTI 2012

- Karlsruhe Institute of Technology + Toyota Technical Institute
- 2,400 annotated dashcam images around Karlsruhe, Germany

Cost Reduction by Design

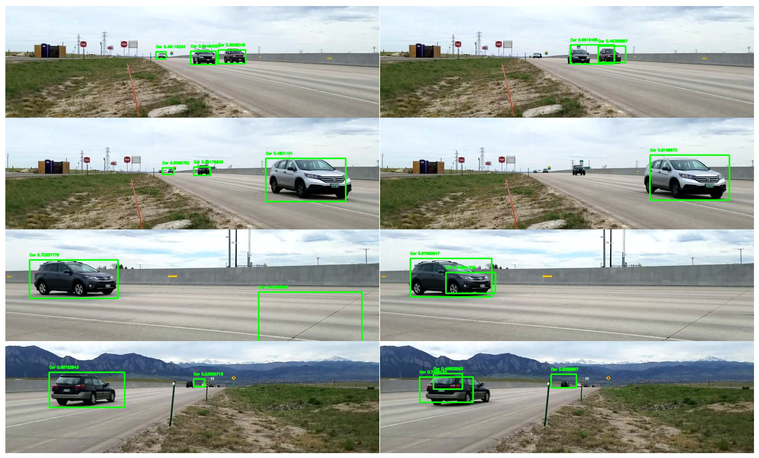
- Original YOLO front-end is conceptually simple: repeated conv / ReLU / max-pool layers
- We replace the front-end CNN with Google’s MobileNet architecture, which is designed for low resource consumption (fewer multiply-accumulates)
- We reduce the number of object classes from 80 to 1, streamlining network output



Results

- Nvidia GTX 960 2GB (training w/ early-stopping, and detection)
- Xeon E5-2650 v3 @ 2.30GHz (detection)
- Full YOLO runs at 1.05 fps CPU, 10 fps GPU (memory-limited?)

MobileNet YOLO	Tiny YOLO
1.04 fps CPU, 23 fps GPU	2.16 fps CPU, 26 fps GPU



Conclusions

- European training data generalizes to Boulder!
  - Blame false negatives on our limited training data (only 2,400 images) and the fact that most examples are parked cars
- MobileNet gives tighter boxes, lower confidence than Tiny YOLO
- MobileNet detection speed comparable to Full YOLO on CPU, but comparable to Tiny YOLO on GPU!
- We could still optimize the detection and NMS IoU thresholds to achieve peak performance on our custom-trained algorithm

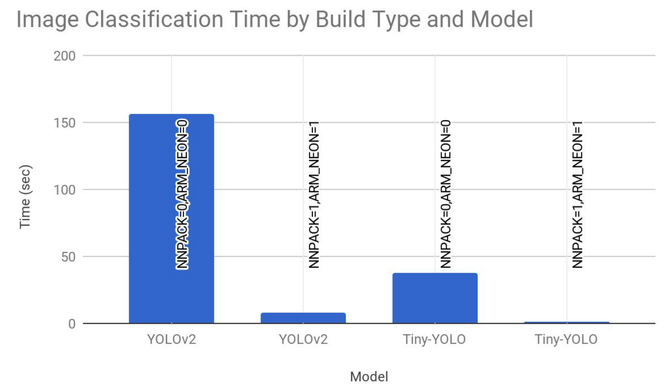
Cost Reduction on Hardware

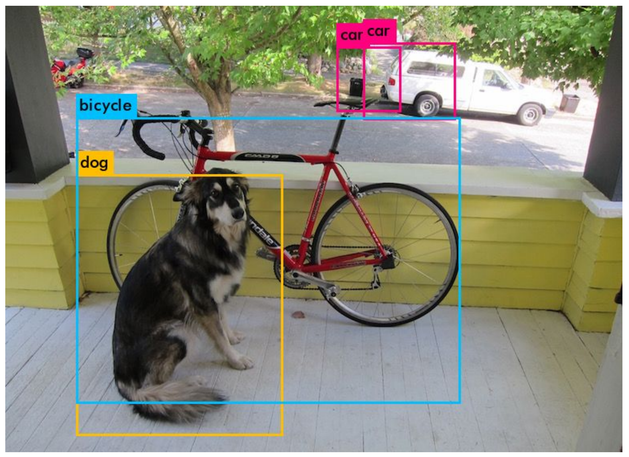
We investigated optimizations that could be done on the Pi in order to make computation faster:

- ARM_NEON: Neon is a library which exposes fast floating point operations on an ARM CPU
- NNPACK: allows for speeding up CNN evaluations by parallelizing computations on a multicore CPU
- Tiny-YOLO: uses a smaller network, decreasing the amount of computation required to perform detection

Results

With all optimizations, we were able to get a 1.3 sec classification time on the Pi! This over a 10x improvement versus the default implementation. But this still means we’re operating at 0.9 fps.





Tiny-YOLOv2 result on sample image