



Tiny YOLO V1 – 32 bit Vs 16 bit comparison

## Purpose of experiment

- To reduce the execution speed of Tiny YOLO V1 algorithm in FlightKit GPU
- Currently we're using 32 bit floating point numbers for executing Tiny YOLO V1 algorithm
- Some OpenCL devices support the **cl\_khr\_fp16** extension, thus reducing storage and bandwidth requirements by a factor 2 compared to 32 bit floating-point operations.

## Float 16 support

**Experiment hardware:** Qualcomm Adreno 330 GPU (OpenCL 1.1)

Available GPU devices		OpenCL 16bit support
Nvidia Geforce GTX 1080 Ti (PC GPU)		X
Qualcomm Adreno 330 (FlightKit GPU)		○

## Experiment output

**Test images:** Sample 100 images from data\_2016\_part1 test data

Evaluation metrics	Expected output	Actual output
Execution speed	Faster than 32 bit computations (< 1.25sec/image)	~1.6 sec/image (scope available for improving speed by tuning)
Accuracy	Same as 32bit computations	No detections (high loss of precision)

## Libraries used

- CLBlast ( <https://github.com/CNugteren/CLBlast> )
  - Used for Float 32 to Float 16 conversion
  - Used for GEMM (General Matrix Multiplication) function
  - Other OpenCL libraries like clBLAS do not support 16 bit computations or do not support OpenCL 1.1

## Step 1

- Read YOLO network weights (32 bit floating point numbers)
- Convert all weights & biases to 16 bit floating point numbers using CLBlast libraries **FloatToHalf()** API function

## Step 2

- Read image from camera/hard disk (32 bit floating point numbers)
- Convert image pixel values to 16 bit floating point numbers using CLBlast libraries **FloatToHalf()** API function

## Step 3

- Pass 16 bit image pixel values through YOLO network's layers with 16bit weights
- Get output - 16 bit class probabilities(confidence) & bounding box co-ordinates
- Convert to 32 bit (optional, using **HalfToFloat()** )

## 32 bit vs 16 bit output comparison

Input image: img\_00839\_1.jpg

Input image		Conv 1 output		Conv 2 output		Conv 3 output		Conv 4 output		Conv 5 output	
32 bit	16 bit	32 bit	16 bit	32 bit	16 bit	32 bit	16 bit	32 bit	16 bit	32 bit	16 bit
0.019608	0.019608	-0.573347	-0.572754	3.583147	3.441406	-1.336501	-1.281258	-0.031947	-0.021484	-0.299501	-0.284180
0.019028	0.019038	-0.621469	-0.620605	4.549234	4.433594	-2.353777	-2.232422	-0.795361	-0.699219	-0.101333	-0.189087
0.015686	0.015686	-0.593348	-0.592285	4.539043	4.425781	-2.313577	-2.191406	-0.917792	-0.816406	-0.374811	-0.362305
0.021317	0.021319	-0.563071	-0.562500	4.514066	4.398438	-2.311015	-2.177734	-0.955555	-0.839844	-0.576911	-0.506348
0.019608	0.019608	-0.608975	-0.607910	4.443643	4.351562	-2.178568	-2.076172	-0.705223	-0.626953	-0.773217	-0.654297
0.019608	0.019608	-0.633424	-0.632324	4.368945	4.265625	-1.856844	-1.765625	-0.491296	-0.452637	-0.655567	-0.560547
0.023529	0.023529	-0.707340	-0.706055	4.298735	4.195312	-1.454983	-1.367188	-0.266636	-0.258057	0.476410	-0.022644
0.019608	0.019608	-0.694428	-0.693359	3.705684	3.619141	-0.947336	-0.893555	-0.523627	-0.468262	0.195930	-0.057800
0.021515	0.021529	-0.693216	-0.691895	2.308382	2.263672	-0.552199	-0.523438	-0.586153	-0.523438	-0.394289	-0.338623
0.028366	0.028372	-0.675384	-0.674316	0.884204	0.896484	-0.398674	-0.374023	-0.392023	-0.354248	-0.560229	-0.447998
0.046326	0.046339										

✖ Output precision loss increases as the number of layers increase

Conv 6 output		Conv 7 output		Conv 8 output		Output	
32 bit	16 bit	32 bit	16 bit	32 bit	16 bit	32 bit	16 bit
-0.371738	-0.274902	2.359346	-0.076904	0.064048	0.132080	FL: 17%	No object detected
-0.444211	-0.286865	-0.865050	-0.342773	-0.001743	0.076172	FL: 22%	
-0.318129	-0.231812	-0.590155	-0.163818	-0.011714	0.092041	FL: 11%	
-0.097037	-0.112183	-0.711898	-0.270508	-0.019095	0.072998	FL: 12%	
-0.177470	-0.139282	-0.400125	-0.183105	-0.031992	0.108887	FL: 43%	
-0.443896	-0.287109	-0.188685	-0.139648	-0.024470	0.118408	FM: 13%	
-0.504550	-0.324951	-0.204270	-0.200928	-0.025531	0.055664		
-0.200541	-0.147461	5.234834	0.664062	0.495687	0.175049		
-0.135773	-0.119873	-0.514025	-0.219849	0.138233	0.115967		
-0.118110	-0.118164	6.415565	1.085938	-0.003259	0.160645		

## Functions that affect precision

- Gemm (Convolution layer, connected layer)
- Activation (Convolution layer, connected layer)
- Normalize (Batchnorm layer)
- Scale bias (Batchnorm layer)
- ....

## Functions that does not affect precision

- Im2col (Convolution layer)
- Maxpooling (Max pool layer)

**From NVIDIA developer forum:**

<https://devtalk.nvidia.com/default/topic/990426/jetson-tx1/tensorrt-yolo-inference-error/>



Hi,

We found YOLO is pretty sensitive to the precision and fp16 mode will slightly lower the output precision. If you want fp16 acceleration, it's recommended to train YOLO model directly on the fp16 mode.

Thanks.

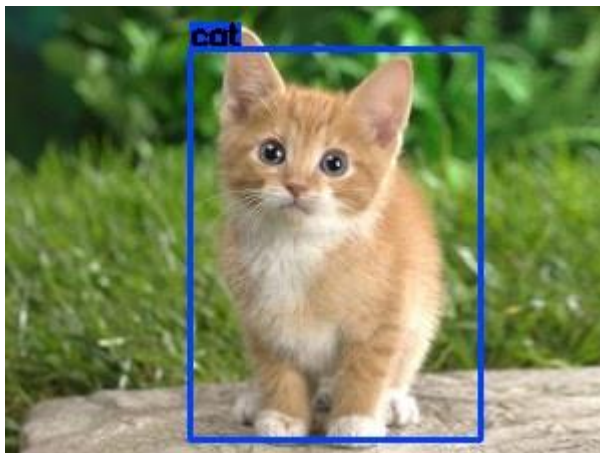
Posted 10/02/2017 06:25 AM

#12

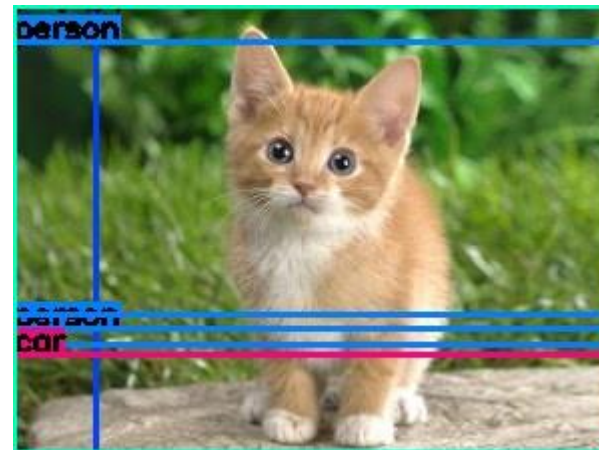
**Other 16 bit YOLO implementations:**

<https://github.com/TLESORT/YOLO-TensorRT-GIE-/>

32 bit output



16 bit output



### Experiment summary:

- Float 16 computations reduces the accuracy of YOLO
- YOLO network is very sensitive to floating point precision change
- Loss in output precision increases as the number of layers increase
- At the moment, only CLBlast support both 16bit computation & OpenCL 1.1 . So, no other libraries available to compare performance & output.

### Points to consider for future:

- Float 16 can be used to reduce the size of the model (Save 16 bit model, but computations done in 32 bit)
  - 32 bit Tiny YOLO V1 model = 86.3 MB
  - 16 bit Tiny YOLO V1 model = 43.1 MB