QuantYOLO: A High-Throughput and Power-Efficient Object Detection Network for Resource and Power Constrained UAVs

Muhammad Gohar Javed[1], Minahil Raza[1], Muhammad Mohsin Ghaffar[2], Christian Weis[2], Faisal Shafait[1,3], Norbert Wehn[2] and Muhammad Shahzad[1,3,4]

[1] *School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Pakistan*
[2] *Microelectronic Systems Design Research Group, Technische Universität Kaiserslautern, Germany*
[3] *Deep Learning Laboratory, National Center for Artificial Intelligence (NCAI), Pakistan*
[4] *Data Science in Earth Observation, Department of Aerospace and Geodesy, TU Munich, Germany*
{*mjaved.bee16seecs, miraza.bee16seecs, faisal.shafait, muhammad.shehzad*}*@seecs.edu.pk*
{*ghaffar, weis, wehn*}*@eit.uni-kl.de*

Abstract—**Convolutional Neural Networks (CNNs) are producing state-of-the-art results in the object detection field. However, deep topologies of CNN are computationally intensive and typically require excessive resources (i.e. high-end GPUs), which hinder their deployment on resource and power constrained UAVs. In this work, we present a high-throughput and power efficient quantized object detection network, QuantYOLO, which is based on the Tiny-YOLOv2 topology. We conduct a detailed exploration of precision and filter pruning vs. accuracy, throughput and power consumption trade-off for the object detection task. As a result of these explorations, we select a network with binarized weights and 4-bit activations (except the output layer), which is 21.8× smaller than the Tiny-YOLOv2 achieving a mean Average Precision (mAP) of 51.5% on the PASCAL-VOC dataset. Finally, we present an FPGA based accelerator, which achieves 1.6× higher throughput (FPS) and is 3.1× more power efficient as compared to prior FPGA architectures.**

Index Terms—**Quantized Convolutional Neural Networks, Object Detection, Pruning, Depthwise Separable Convolutions, Real-Time and Power-Efficient Architecture.**

I. Introduction

State-of-the-art Deep Learning (DL) based object detection networks, such as You Only Look Once (YOLO), Region-based Convolutional Neural Networks (R-CNN) and Single Shot Detector (SSD), are based on Convolutional Neural Networks (CNNs) [1]–[3], and are widely used in search and rescue, infrastructure/crop/forest inspection, and surveillance domains. There has been an increasing interest at the edge-inference of these object detection algorithms on Unmanned Aerial Vehicles (UAVs) to achieve real-time performance. Although Graphics Processing Units (GPUs) can achieve real-time performance, the high power consumption of GPUs is a bottleneck in the deployment of these algorithms at the edge.

Although, the detection accuracy of DL-based object detection algorithms has increased considerably during the last decade, the computation complexity and memory requirements of these algorithms have also grown. As a result, inference implementations are unable to fulfil the stringent power and throughput requirements for their embedded deployment. Therefore, model compression techniques such as quantization [4], depthwise separable convolutions [5] and pruning [6] play a critical role in reducing the computational complexity and memory requirements without effecting the accuracy. Considering all the mentioned limitations from software and architecture perspective, Field-Programmable Gate Arrays (FPGAs) are ideal candidates for the deployment of object detection algorithms due to three reasons. First, they allow customized hardware architecture with bit-level parallelism that results in high energy efficiency and throughput. Second, FPGAs can use custom memory hierarchies, which are critical for efficient data transfers and processing. Third, FPGAs support customized precision data types, such as 1-bit weight and 4-bit activation, which is considered for the presented architecture. Hence, we selected the Xilinx Zynq ZC706 FPGA development board with XC7Z045 device as our deployment platform.

The design of CNN object detector accelerators has become an active area of research in recent years. In [7], the authors proposed scalable convolutional blocks for real-time object detection. In [8], the authors conducted experiments with CNN-based object detectors on Xilinx FPGAs. The proposed architecture was unable to achieve real-time inference results. The work in [9] presents experiments with multiple object detection networks on a Xilinx Zynq 7000 SoC device. [10], [11] explore low bit-width quantization for object detection. In both of these works, quantization results in a tremendous loss to the network accuracy. YOLO-LITE [12] and Mixed YOLO-v3-LITE [13] explore architectural modifications in the object detectors, YOLO and Tiny-YOLO. In contrast to previous works, we present a model, QuantYOLO, which combines different compression techniques i.e. quantization, pruning and, depthwise separable convolution without significant loss of detection accuracy. Furthermore, we also present a hardware architecture to efficiently map the inference on an FPGA for low power and real-time object detection. The major contributions of our work are summarized as follows:

- We cross-correlate the gain in throughput with the loss in accuracy for a range of quantized bit-widths for weights and activations to achieve an optimal throughput-accuracy trade-off. Furthermore, We investigate the efficacy of combining pruning and depthwise separable convolutions with quantization for our network. Finally, we present an optimal trade-off between compression techniques and accuracy, throughput, and power consumption for deployment of the object detection networks on resource and power constrained UAVs.
- We present an FPGA architecture for real-time object detection, QuantYOLO, with 1-bit weights and 4-bit activations (except the output layer which uses 32-bit activations). The proposed architecture achieves 25.29 FPS, which is a 1.6× improvement over previous FPGA-based architectures [8], [14]–[20]. Furthermore, the power efficiency of the architecture is 13.17 FPS/W, which is 3.1× improvement.
- The results of the Runway-Det dataset illustrate that our architecture is 7.78× energy efficient, while accomplishing 3.7× higher FPS as compared to Nvidia Jetson Nano embedded GPU.

The paper is structured as follows: in Section II we explain the background of the object detection and quantized CNNs. In Section III we describe the training approach, ablation study and experiments for designing our network topology. In Section IV we describe the architecture for inference on FPGA. Section V discusses our results. Finally, Section VI concludes our work.

II. Background

A. Object Detection using CNNs

YOLO [1] is a state-of-the-art object detection network. It redefines object detection as a single regression problem. The image is divided into a grid of $S \times S$ cells. Each grid cell detects an object and predicts B bounding boxes and the corresponding confidence scores, when the center of an object lies in that cell. B is the number of anchor boxes associated with every grid cell. Each bounding box prediction has 5 components: $(x, y, w, h, confidence\ score)$. (x, y) are coordinates of the centre of the box while (w, h) are its dimensions. The $confidence\ score$ is the probability of presence of an object in the bounding box. For class probabilities, a vector of length C $(number\ of\ classes)$ is associated with each grid cell. Consequently, a three-dimensional tensor with dimensions $S \times S \times B(5 + C)$ is the output. YOLO employs Non-Maximal Suppression (NMS) to avoid predicting multiple bounding boxes for the same object.

B. Neural Networks Compression

Deep neural networks have millions of parameters (i.e. weights and activations) and require billions of operations to complete their task. Such as, a neural network (NN) topology VGG-16 [21] has a model size of 560 MB and requires up to 15.8 billion computations. Due to this reason, NN compression methods are vital in reducing the size of the model and making it suitable for porting to an embedded device. However, a balanced trade-off between these compression techniques is important due to their impact on the accuracy of the algorithm.

Quantization plays a key role in achieving comparative levels of accuracy as compared to full precision (32-bit floating point) networks. The 8-bit quantized CNNs have shown minimal accuracy reduction as compared to full precision networks. Aggressive quanitzation i.e. Binary Neural Networks (BNNs), in which both weights and activations are constrained to +1 and -1 or 0 and 1 respectively. While BNNs give good results for simpler networks, they can deteriorate accuracy for deep networks such as object detection algorithms. Therefore, Quantized Neural Networks (QNNs), which use fixed point representation for weights and activations are preferred for such tasks. QNNs provide more flexibility in choosing bit-widths for the network. [22], [23] explore methods to train BNNs and QNNs. In this work, we have explored with different bit-widths for weights and activations to select optimal bit-width for the object detection task.

Pruning helps in reducing the latency as well as the memory footprint of CNNs through the removal of parameters (i.e. weights and activations) from the network based on sparsity. It works on the basis of a binary criteria to decide which weights are excluded from a network. The excluded elements are trimmed from the model, their values are set to zero and are not updated during back propagation.

A depthwise separable convolution layer is a combination of two layers; a depthwise convolution followed by a pointwise convolution. A depthwise convolution is a spatial convolution performed independently over each channel of an input. A pointwise convolution is a 1x1 convolution which operates over the depth and changes the number of channels instead of other dimensions. It projects the channels output by the depthwise convolution onto a new channel space. Overall, these convolutions have lesser parameters and operations as compared to the standard convolutions.

III. Training

For our experiments, we modify the Lightnet [24] framework to support quantization and pruning. The experiments are evaluated on two datasets shown in Table I. We use the PASCAL-VOC 2007+2012 [25] dataset for training and PASCAL-VOC 2007 for validation, similar to the approach used to train YOLO. In addition, we train and evaluate our results on a runway detection dataset, Runway-Det. This dataset consists of images of runways collected from Google Earth [26]. The images contain only one object (runways) for autonomous landing of UAVs. Selected samples of this dataset are shown in the Fig. 1. The hyperparameters used for training are presented in Table II. We trained our model on PASCAL-VOC and then used transfer learning to fine-tune the network on the Runway-Det dataset.

TABLE I
DETAILS OF DATASETS USED IN THIS WORK

Dataset	Number of Classes	Training Images	Testing Images
PASCAL-VOC	20	16,511	4,952
Runway-Det	1	4727	1,000

TABLE II
HYPERPARAMETERS USED FOR TRAINING OF THE NETWORK

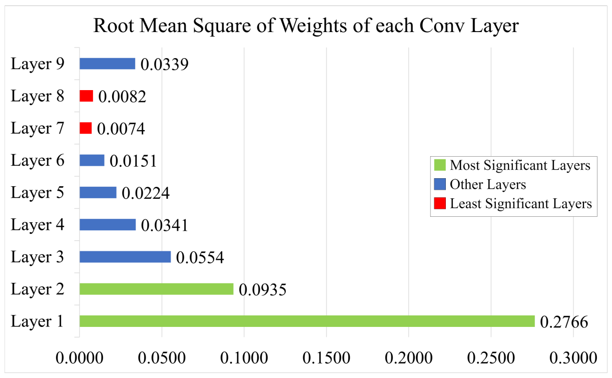
Batch size	64
Mini batch size	8
Number of anchor boxes	5
Optimization algorithm	Adam [27]
Learning Rate	$0.0025/Batchsize$
Confidence threshold	0.25
NMS threshold	0.5
Input Image Resolution	$416 \times 416 \times 3$

A. Ablation Study

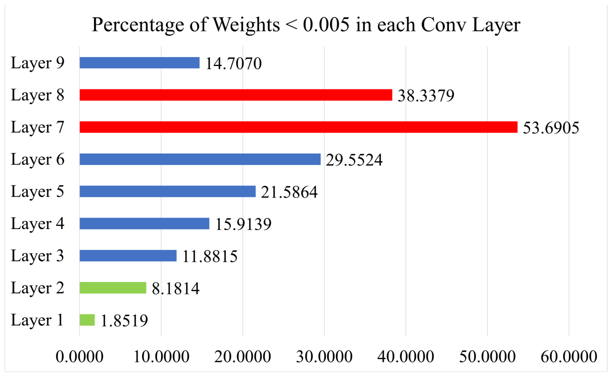
We selected the Tiny-YOLOv2 [28] topology as our baseline model since it is smallest and fastest of the YOLO object detectors. Later versions of Tiny-YOLO, as described on the Darknet website [29], focus on improving accuracy which results in substantial increase of computational costs. Tiny-YOLOv2 is a single forward-pass object detector with only 9 layers. Hence, we found it to be at an ideal spot on the accuracy and throughput trade-off. We acquired the pre-trained weights of Tiny-YOLOv2, on PASCAL-VOC, from [30], and performed an ablation study to compress the network. The significance of each layer for the output is quantified by the root mean square (RMS) of weights and the percentage of weights (w) close to zero i.e, $-0.005 < w < 0.005$, in each Convolution (CONV) layer of the pre-trained network, see Fig. 2. This analysis is based on exploiting the sparsity in the network.

This analysis suggests that layers 1 and 2 of Tiny-YOLOv2 are highly significant, while layers 7 and 8 have a considerable number of weights close to zero. A high ablation ratio of 50% for layers 7 and 8 results in a slight drop of mAP. To compensate for this, the number of filters in the layers 1 and 2 are increased. Our experiments (see Table III) indicate that the *Config 4*, with 50% filter ablation from layers 7 and 8 followed by a 100% increase in layer 2 filters, produces the best performing network. It achieves $2.49\times$ filter reduction as compared to Tiny-YOLOv2 with a drop of only 1% in mAP. We chose this configuration to perform further network compression as described in subsequent sections.

Fig. 1. Sample images from the Runway-Det Dataset.



(a)



(b)

Fig. 2. (a) Comparison of Root Mean Square of weights for CONV layers (b) Comparison of percentage of weights with values < 0.005 for CONV layers

TABLE III
ABLATION STUDY EXPERIMENTS

Configuration	C_L[1]				mAP	No. of 3x3x1 filers
	C_1	C_2	C_7	C_8		
Tiny YOLOv2[2] [28]	16	32	1024	1024	57.1%	1,747,504
Config 1	16	32	512	512	54.9%	698,928
Config 2	32	64	512	512	56.4%	702,560
Config 3	32	32	512	512	55.3%	699,488
Config 4	16	64	512	512	56.2%	701,488

[1] C_L represents number of channels C in layer L.

[2] The configuration of Tiny YOLOv2 and its mAP on PASCAL-VOC were taken from [30]

B. Quantization

We use the same quantization approach, which is presented in [22], [23]. We adopt Eq. 1a as a binarization function for weights. We scale the binarized weights by their channel-wise means. For multi-bit activations, we use the function in Eq. 1b, where x is the input real number, $x \in [0, 1]$, which is quantized to k bits to produce output Q. We further quantize the gradients to 16-bits to accelerate training using Eq. 1c, where g_L is the full-precision gradient tensor for layer L, $max()$ computes the maximum value of its input, Q_k is the function defined in Eq. 1b, and g_L^k is the quantized gradient tensor for layer L quantized to k bits.

$$f(x) = \begin{cases} -1, & x < 0 \\ +1, & x \geq 0 \end{cases} \tag{1a}$$

$$Q_k(x, k) = \frac{round(x \times (2^k - 1))}{2^k - 1} \tag{1b}$$

$$g_L^k = 2max(|g_L|) \times [Q_k(\frac{g_L}{2max(|g_L|)} + \frac{1}{2}) - \frac{1}{2}] \tag{1c}$$

We name the networks as WxAy where x and y are the number of bits for weights and activations respectively. A series of experiments reveal that activations of last layer (the output) could not be quantized because of the nature of post-processing in YOLO algorithm. Following are the experiments we performed with different bit-widths for weights and activations of our network.

- **W1A4-PartialQuant-1**: The first and last three layers are not quantized while the middle ones are quantized to W1A4.
- **W1A4-PartialQuant-2**: All the layers are fully binarized (W1A4) except for input and output layers, which uses 32-bit precision.
- **W1A4-FullQuant**: The input image is quantized to 8-bits. Weights of all layers are binarized, while all activations except for last layer are quantized to 4-bits.
- **W1A2-Quant**: The bit-width of the hidden layer activations in W1A4-FullQuant is changed to 2-bits.

C. Depthwise Convolutions and Pruning

We conduct experiments to evaluate the suitability of depthwise separable convolutions and filter pruning for our quantized network. For pruning, we use Frobenius norm-based (FN) [6] and Geometric median-based (GM) [31] filter pruning. The pruning rate is set at 0.1 in our experiments, which translates to the removal of 10% filters with lowest norm values from each layer. Since pruning is based on sparsity in NNs, we could not use high percentages with QNNs. We experimented with the following approaches:

- **Depthwise-FullPrec**: Layers 2 to 7 of our full precision network are replaced with depthwise separable convolution layers.
- **Depthwise-W1A4**: The Depthwise-FullPrec model is quantized to W1A4.
- **FN-Pruned-W1A4**: All layers except the last layer of the W1A4-FullQuant model are pruned using FN as the criteria.
- **GM-Pruned-W1A4**: All layers except the last layer of the W1A4-FullQuant model are pruned on the basis of GM. GM is more reasonable as compared to FN as it searches for a point x' that minimizes the sum of Euclidean distance to a set of n points , $a^1, a^2, ..., a^n$, see Eq. 2a and 2b. Therefore, GM is more robust.

$$x = argminf(x) \tag{2a}$$

where

$$f(x) = \sum_{i \in [1,n]} ||x - a^i||_2 \tag{2b}$$

D. QuantYOLO topology

Our experiments show that the best results are achieved for the W1A4-FullQuant model. Fig. 3 illustrates the topology of the QuantYOLO network. All the layers of the network have been quantized to a smaller bit-width. The input layer in Fig. 3 represents the input images to the model. The last CONV layer (Layer 9) is a pointwise convolution layer with K number of filters, given by Eq. 3.

$$K = B * (5 + C) \tag{3}$$

Where B is the number of anchor boxes and C is the number of classes in the dataset. We use five anchor boxes. Thus, K is equal to 125 and 30 for PASCAL-VOC and Runway-Det respectively.

IV. Architecture for Inference

We used an existing architecture proposed in [32]. The CONV, batch normalization and activation layers of our quantized model are approximated through bit-wise operations. For BNNs, the multiply-and-accumulate (MAC) operations in the CONV layers are represented by the bit-wise XNOR followed by the popcount operation. An XNOR operation, shown in Eq. 4a, yields the same result as multiplication when **x** and **y** are vectors of $\{-1, 1\}$. The bit-wise operation kernel in Eq. 4b is used instead of the MAC operation for 1-bit weights and 4-bit activations. In the popcount operation, number of set bits (bits with value 1) are counted.

$$\mathbf{x} \cdot \mathbf{y} = N - 2 \times popcount(xnor(\mathbf{x}, \mathbf{y})), \quad x_i, y_i \in \{-1, 1\} \; \forall i \tag{4a}$$

$$\mathbf{x} \cdot \mathbf{y} = \sum_{k=0}^{3} 2^k \times popcount[xnor(\, c_0(\mathbf{x}), \, c_k(\mathbf{y}))], \quad c_0(\mathbf{x})_i, \, c_k(\mathbf{y})_i \in \{0, 1\} \; \forall i, k \tag{4b}$$

Xilinx FPGAs have two parts - Processing System (PS) and Programmable Logic (PL). PS consists of ARM cores and is suitable for sequential tasks. PL is the programmable (FPGA) part, which is ideal for parallel tasks. Our NN is partitioned between PL and PS parts. The post-processing steps of QuantYOLO are performed on the PS side owing to their sequential nature, which is accelerated through vectorization. Since CONV layers are the most compute intensive, multiple channels are processed in parallel on the PL side to reduce latency. [32] achieves parallelism through multiple computation units called Processing Engines (PEs) which have

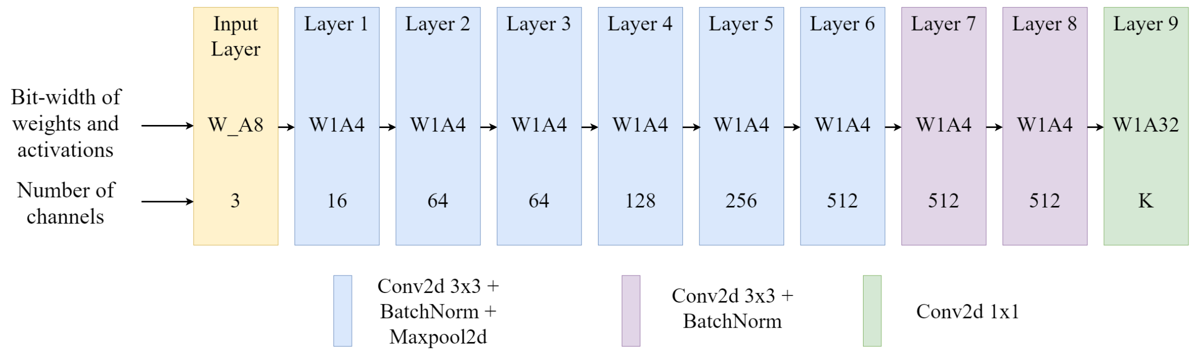


Fig. 3. The QuantYOLO topology.

multiple Single-Instruction-Multiple-Data (SIMD) lanes for high performance. The activation function and batch normalization are implemented as thresholding steps. The thresholds are learnable parameters of the model which are computed during the weight generation step after training. We also use maxpooling layers which are implemented as a separate hardware unit. For the implementation of convolution layers on PL, we experimented with two approaches; Multi-layer Offloading (MO) and Feed Forward Dataflow (FFD). In the FFD implementation, all the CONV layers are implemented on the PL part of the FPGA and the network parameters for all layers are cached in the on-chip memory. This eliminates the need to access DRAM (Dynamic Random Access Memory) in between CONV layers and reduces the memory cost during runtime. However, in case of the the MO implementation, a fixed architecture (only one CONV layer) is implemented on the PL part of the FPGA and the weights and inputs are loaded from DRAM after each CONV layer completes processing. Moreover, the communication overhead between PS and PL should also be kept in mind during the deployment phase. We experiment with following partitions between PS and PL sides:

- **Without HW Offload:** All the layers of the network are implemented on the PS to analyze the network performance without HW offloading.
- **HW Offload with MO:** The first and last layer of the network are implemented on the PS while the rest of the layers are executed on the PL side one by one. We tested this configuration because the first and last layer have 16 and 125 output channels respectively, which make them less compute intensive as compared to rest of the layers.
- **Maximum HW Offload using FFD:** All CONV layers are deployed on the PL side. Only the post processing is performed on the PS side.
- **Vectorization of Post-processing:** Intersection over Union (IoU) calculation and Non-maximal suppression (NMS) are vectorized using matrix operations to increase throughput.
- **Multi-threading:** The previous technique is modified by using multi-thread parallelism for post-processing steps. Multi-threading is done using open source Python libraries.

Fig. 4 shows a high-level view of our final FPGA implementation.

V. Results

To evaluate the performance of the proposed compression techniques and FPGA architecture, we experiment with two datasets, PASCAL-VOC and, Runway-Det as presented in Section III. Two embedded hardware platforms, Xilinx Zynq ZC706 FPGA development board with XC7Z045 chip and, Nvidia Jetson Nano, are explored for embedded inference of the QuantYolo topology. For deployment on the FPGA, the Vivado High Level Synthesis (HLS) tool, which is a part of Vivado Design Suite 2018.2, is used.

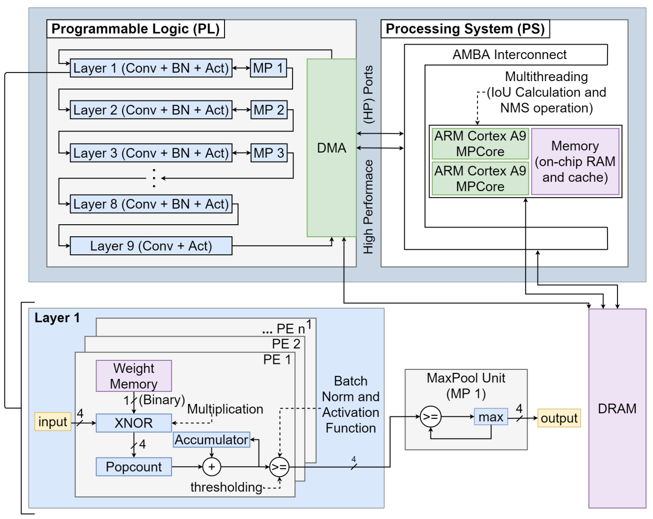


Fig. 4. The CONV, batch normalization, activation and maxpool layers of QuantYOLO are implemented on the PL side, while pre-processing and post processing are implemented on the PS side.

A. *Performance Metrics*

We use the following metrics to evaluate our network on the above mentioned hardware platforms:

1) Mean Average Precision (mAP): It is the mean of the average precision (AP) for all classes in a dataset. The Average Precision (AP) for each class is the mean of precision values for a range of recall values over 0 to 1. This metric measures represents the accuracy of an object detection algorithm.

2) Throughput (Frames Per Second (FPS)): Frames per second is the number of image frames processed in one second. It is a metric to quantify the throughput of an algorithm.

3) Power Consumption: Each hardware platform operates under different hardware setup that exhibits different power consumption due to different peripherals that do not participate in computations but contribute to the overall power consumption. For fair comparison, we compute the consumed power based only on power consumption of components that participate during computation. Following formula is used to calculate the power:

$$Power_{value} = Power_{dynamic} - Power_{idle} \quad (5)$$

where, $Power_{dynamic}$ is the power consumption of the complete system while processing the CNN inference algorithm on the hardware platform and, $Power_{idle}$ is the power consumption of the complete system in the idle state. By using this formula, we minimize the influence of power consumption of the extra hardware and the peripherals.The power has been measured physically using digital wall socket power meter Voltcraft VC-870.

B. *Accuracy comparison based on mAP*

A total of eight experiments are performed for network complexity reduction as described in Section III. Hence, we are searching for an optimal solution to a multi-objective problem, it is considered as a Pareto optimization problem. The Pareto-frontier chart in Fig 5, shows the mAP vs model size trade-off for each of our implementations. It can be seen that W1A4-FullQuant sits on an ideal spot on the Pareto-frontier. It achieves a mAP score of 51.5% with a model size of only 2.75 MB. Although, W1A4-PartialQuant-2 achieves a higher mAP score of 52.7%, the precision of 32-bits for the the input image layer increases the computational complexity of the network, which exponentially scales up resource utilization on the FPGA. Our experiments show that extremely low bit-width quantization for activations (2-bits) results in a significant loss of mAP score for shallow networks. Moreover, for YOLO networks, keeping the output layer at 32-bit precision helps in maintaining the mAP score. Therefore, on basis of obtained results and our hardware critical specifications, we selected the W1A4-FullQuant model for FPGA implementation. The mAP score, model size and number of parameters by combining different compression techniques are compared in Table IV along with the previous works.

Our QuantYOLO model achieves an impressive mAP score as compared to previous works with a compact network of only 9 layers and 2.75 MB of size. We achieve a 21.8× reduction

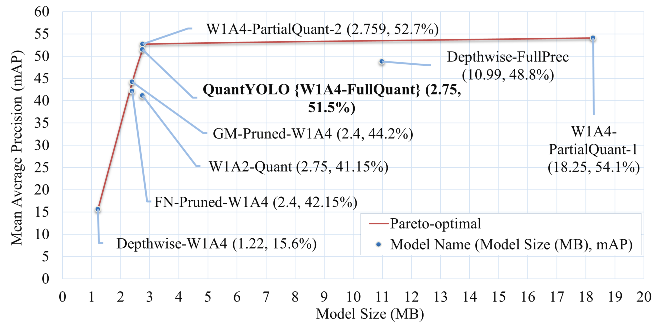


Fig. 5. Pareto-frontier for mAP against model size for the PASCAL-VOC dataset.

in model size of Tiny-YOLOv2 with minimal loss in mAP. When compared to YOLO-LITE [12], we achieve 17.73% higher mAP with a comparable model size. For depthwise convolution, we outperform MobileNet v2 implementation in [13] in terms of mAP with less parameters and a smaller model size. When compared to previous quantization works in [10], [33], we achieve higher mAP with lesser number of parameters and smaller model size. Combining pruning with quantization does result in a drop in mAP. However, even with a pruned network, we achieve 44.2% mAP. Experiments with pruning indicate that GM-based pruning achieved better results for binarized networks as compared to the FN-based.

C. *Throughput optimization on FPGA*

Fig. 6 shows the throughput comparison of different hardware implementation approaches presented in Section IV. As a result of these explorations, we achieve a final throughput of 21.7 FPS and 25.29 FPS with and without accounting for the post-processing steps respectively.

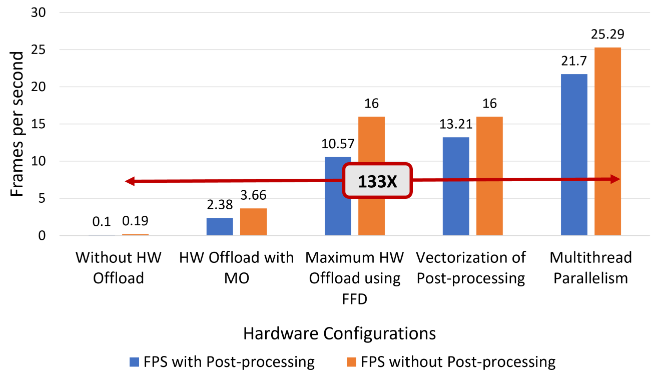


Fig. 6. FPS results for different hardware configurations on the ZC706 FPGA development board.

D. *Comparison With Previous Works*

Table V presents the detailed comparison of our architecture with existing architectures for object detection. Furthermore, we also compare our architecture with the Nvidia Jetson Nano embedded GPU. All values of FPS and power consumption for our work are calculated by taking mean over 5 individual

TABLE IV
mAP COMPARISON ON PASCAL-VOC (NETWORK COMPRESSION TECHNIQUES USED BY EACH MODEL ARE GIVEN IN TABLE NOTES)

	Existing Literature						**This Work**		
Model	Tiny-YOLO v2 [28]*	YOLO-LITE [12][1]	Mixed YOLOv3-LITE [13][1]	MobileNet V2-YOLO v3 [13][3]	Quantized Tiny-YOLO v2 [10][2]	Resnet18 Faster RCNN [33][2]	QuantYOLO [W1A4-FullQuant][2,4]	Depthwise-FullPrec[3,4]	GM-Pruned-W1A4[2,4,5]
Weights Precision [bits]	32	32	32	32	1	1	1	32	1
Activations Precision [bits]	32	32	32	32	4	16	4	32	4
No. of CONV Layers	9	7	34	76	9	17	9	9	9
Model Size [MB]	60	2.3	18.8	93.5	4.74	-	2.75	10.99	2.4
No. of Parameters [M]	15.86	-	5.089	23.27	15.86	10.999	6.38	2.9	5.45
mAP	57.1%	33.77%	48.25%	13.26%	49.12%	47.0%	51.5%	48.8%	44.2%

* The numbers for Tiny-YOLOv2 are taken from [30].
[1] Reduced number of layers.
[2] Network quantization.
[3] Depthwise separable convolutions.
[4] Layer-oriented filter pruning.
[5] Filter pruning (GM or FN).

TABLE V
COMPARISON WITH PREVIOUS WORKS AND NVIDIA JETSON NANO

	Platform	Precision	CONV Layers	Latency $[ms]$	FPS	Power $[W]$	Power eff. $[FPS/W]$
[19]	Intel Cyclone V	W16A16	9	339	2.95[1]	-	-
[20]	Intel Arria 10 GX 1150	W8A16	21	61.4	16.29[1]	40	0.41
[8]	Xilinx PYNQ	W32A32	24	-	3.98	-	-
[14]	Xilinx PYNQ	W1A1	9	-	11.1	2.61	4.25
[15]	Xilinx PYNQ	W8A8	16	202	4.95[1]	2.32	2.13
[16]	Xilinx Zynq 7030	W8A8	9	70.22	14.24	-	-
[17]	Xilinx ZC706	W32A32	24	744	1.34[1]	**1.17**	1.15
[18]	Xilinx KU115	W8A8	24	65	15.4[1]	13	1.18
This Work	Nvidia Jetson Nano	W32A32	9	153.14	6.53	4.06	1.61
This Work	Xilinx ZC706	W1A4	9	**39.54**	**25.29**	1.92	**13.17**

[1] FPS not reported, calculated using $1/latency$, considering latency= time taken to process one image.

TABLE VI
COMPARISON WITH EMBEDDED GPU FOR THE RUNWAY-DET

Network	**Platform**	**Average Precision**	**FPS**	**Power** $[W]$	**Power eff.** $[FPS/W]$
Full-precision (32-bit)	Jetson Nano	73.2%	6.92	4.02	1.72
QuantYOLO (W1A4)	ZC706	70.1%	25.42	1.90	13.38

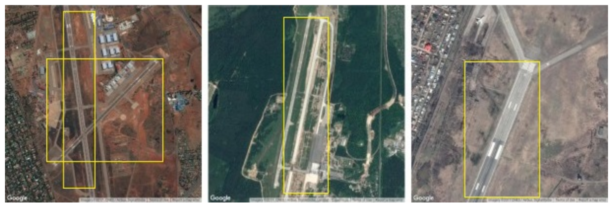


Fig. 7. Bounding box predictions for the Runway-Det Dataset.

trials of inference over 105 samples. The power efficiency is calculated by using the formula, $Power\ eff = \frac{FPS}{Power}$.

The FPGA architecture achieves a throughput of 25.29 FPS as compared 6.53 FPS of the Nvidia Jetson Nano. Furthermore, the power consumption of the FPGA architecture is 1.92 W, which is 2.1× better than the embedded GPU. In terms of power efficiency, FPGA is 8.2× better than the GPU implementation. On the basis of these results, we conclude that the FPGA is an ideal processing system for porting DL models on the power-constrained UAVs.

In comparison to the previous works [8], [14]–[20] (see Table V), our work achieves 1.6× higher throughput and is 3.1× energy efficient. On basis of these results, we conclude that the object detection algorithms can greatly benefit from the NN compression techniques during the inference phase. Using full-precision (FP32) for the inference is of a negligible benefit and only results in increased latency, hardware cost and power consumption.

E. Results on the Runway-Det dataset

In addition to the PASCAL-VOC benchmark, the QuantYOLO model is used for the Runway-Det dataset. Fig 7 shows the inference results of the selected samples from the test dataset. The FPGA and GPU implementation results are presented in Table VI. It can be seen from the results that the proposed architecture achieves 3.7× higher FPS and, consumes 2.12× less power than the embedded GPU. Furthermore, our architecture is 7.78× more power efficient than the corresponding GPU implementation. From these results, it is evident that the proposed object detection topology and architecture are extremely favourable for low power embedded devices such as drones and UAVs.

VI. Conclusion

In this paper, we present, QuantYOLO, a high throughput and, power efficient object detection NN accelerator for resource and power constrained UAVs. We lower the complexity of the Tiny-YOLOv2 model by taking advantage of NN compression techniques such as quantization, filter pruning and, depthwise separable convolutions. As a result of these explorations, the presented model that has 9 layers and, 2.75 MB model size with precision of 1-bit for weights and 4-bit for activations. Our model achieves a mAP score of 51.5% on PASCAL-VOC benchmark and 70.1% average precision on the Runway-Det dataset. In comparison to the previous works, our FPGA accelerator achieves 25.29 FPS while consuming 1.92 W power, which translates to a 1.6× and 3.1× improvement in terms of FPS and power efficiency respectively. Furthermore, for the Runway-Det dataset, the presented architecture is 7.78× power efficient as compared to the Nvidia Jetson Nano embedded GPU. In the future, we aim to extend this work to perform object tracking.

Acknowledgement

Part of this research is supported by the German Academic Exchange Service (DAAD). We also thank the anonymous reviewers for their valuable suggestions.

References

[1] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[3] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

[4] Y. Guo, "A survey on methods and theories of quantized neural networks," *CoRR*, vol. abs/1808.04752, 2018. [Online]. Available: http://arxiv.org/abs/1808.04752

[5] V. Vanhoucke, "Learning visual representations at scale," *International Conference on Learning Representations (ICLR)*, 2014.

[6] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *CoRR*, vol. abs/1608.08710, 2016. [Online]. Available: http://arxiv.org/abs/1608.08710

[7] B.-G. Han, J.-G. Lee, K.-T. Lim, and D.-H. Choi, "Design of a scalable and fast yolo for edge-computing devices," *Sensors*, vol. 20, no. 23, p. 6779, Nov 2020.

[8] S. P. Kaarmukilan, S. Poddar, and A. Thomas K., "Fpga based deep learning models for object detection and recognition comparison of object detection comparison of object detection models using fpga," in *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, 2020, pp. 471–474.

[9] R. Yerne, B. Patil, and Y. Ghorpade, "A review paper on object detection using zynq-7000 fpga for an embedded applications," *International Research Journal of Engineering and Technology*, vol. 07, no. 01, Jan 2020. [Online]. Available: https://www.irjet.net/archives/V7/i1/IRJET-V7I189.pdf

[10] D. T. Nguyen, T. N. Nguyen, H. Kim, and H. Lee, "A high-throughput and power-efficient fpga implementation of yolo cnn for object detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1861–1873, 2019.

[11] R. Li, Y. Wang, F. Liang, H. Qin, J. Yan, and R. Fan, "Fully quantized network for object detection," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2805–2814.

[12] J. Pedoeem and R. Huang, "YOLO-LITE: A real-time object detection algorithm optimized for non-gpu computers," *CoRR*, vol. abs/1811.05588, 2018. [Online]. Available: http://arxiv.org/abs/1811.05588

[13] H. Zhao, Y. Zhou, L. Zhang, Y. Peng, X. Hu, H. Peng, and X. Cai, "Mixed yolov3-lite: A lightweight real-time object detection method," *Sensors (Basel, Switzerland)*, vol. 20, no. 7, Mar 2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7180807/

[14] L. Yang, Z. He, and D. Fan, "Binarized depthwise separable neural network for object tracking in fpga," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 347–350. [Online]. Available: https://doi.org/10.1145/3299874.3318034

[15] Y.-C. Ling, H.-H. Chin, H.-I. Wu, and R.-S. Tsay, "Designing a compact convolutional neural network processor on embedded fpgas," in *2020 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*, 2020, pp. 1–7.

[16] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Song Yao, Song Han, Yu Wang, and Huazhong Yang, "From model to fpga: Software-hardware co-design for efficient neural network acceleration," in *2016 IEEE Hot Chips 28 Symposium (HCS)*, 2016, pp. 1–27.

[17] R. Zhao, X. Niu, Y. Wu, W. Luk, and Q. Liu, "Optimizing cnn-based object detection algorithms on embedded fpga platforms," in *ARC*, 2017.

[18] J. Yu, Y. Hu, X. Ning, J. Qiu, K. Guo, Y. Wang, and H. Yang, "Instruction driven cross-layer cnn accelerator with winograd transformation on fpga," *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017.

[19] Y. J. Wai, Z. M. Yussof, S. I. Salim, and L. Chuan, "Fixed point implementation of tiny-yolo-v2 using opencl on fpga," 2018.

[20] Y. Ma, T. Zheng, Y. Cao, S. Vrudhula, and J.-s. Seo, "Algorithm-hardware co-design of single shot detector for fast object detection on fpgas," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, Nov 2018, p. 1–8. [Online]. Available: https://dl.acm.org/doi/10.1145/3240765.3240775

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556 [cs]*, Apr 2015, arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556

[22] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *CoRR*, vol. abs/1603.05279, 2016. [Online]. Available: http://arxiv.org/abs/1603.05279

[23] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *CoRR*, vol. abs/1606.06160, 2016. [Online]. Available: http://arxiv.org/abs/1606.06160

[24] T. Ophoff, "Lightnet: Building blocks to recreate darknet networks in pytorch," https://gitlab.com/EAVISE/lightnet, 2018.

[25] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, p. 303–338, Jun 2010.

[26] "Google earth." [Online]. Available: https://earth.google.com/web/

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[28] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[29] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[30] "Yolov2: Real-time object detection." [Online]. Available: https://pjreddie.com/darknet/yolov2/

[31] Y. He, P. Liu, Z. Wang, and Y. Yang, "Pruning filter via geometric median for deep convolutional neural networks acceleration," *CoRR*, vol. abs/1811.00250, 2018. [Online]. Available: http://arxiv.org/abs/1811.00250

[32] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. ACM, 2017, pp. 65–74.

[33] Y. Wei, X. Pan, H. Qin, and J. Yan, "Quantization mimic: Towards very tiny cnn for object detection," in *ECCV*, 2018.