An Efficient Real-Time Object Detection Framework on Resource-Constricted Hardware Devices via Software and Hardware Co-design (Invited Paper)

Mingshuo Liu*, Shiyi Luo*, Kevin Han*, Bo Yuan†, Ronald F. DeMara‡, Yu Bai*
* California State University, Fullerton
{liumingshuo, luoshiyi, khchocosj}@csu.fullerton.edu, ybai@fullerton.edu
†Rutgers University
bo.yuan@soe.rutgers.edu
‡University of Central Florida
Ronald.Demara@ucf.edu

***Abstract*—The fast development of object detection techniques has attracted attention to developing efficient Deep Neural Networks (DNNs). However, the current state-of-the-art DNN models can not provide a balanced solution among accuracy, speed, and model size. This paper proposes an efficient real-time object detection framework on resource-constricted hardware devices through hardware and software co-design. The Tensor Train (TT) decomposition is proposed for compressing the YOLOv5 model. By unitizing the unique characteristics given by the TT decomposition, we develop an efficient hardware accelerator based on FPGA devices. Experimental results show that the proposed method can significantly reduce the model size and improve the execution time.**

***Index Terms*—Deep learning, FPGA, Tensor Train, Energy Efficiency**

I. Introduction

Nowadays, from companies to academics, researchers across the world are interested in developing Deep Neural Networks (DNNs) due to their incredible feats in various applications, such as image recognition, playing complex games, and large-scale information retrieval such as web search. Among them, object detection as the major computer vision task has been attracted with more research efforts in the computer vision field, including event detection, image segmentation, action detection, and image annotation. These tasks are widely employed in autonomous robotics, UAV obstacle avoidance, and robot vision that demand high accuracy and low latency DNN running on resource-limited electronic devices.

However, the current DNN can achieve a strong representation and consume more resources simultaneously. For example, the ResNet consists of 152 layers with more than 60 million parameters [1]. Consequently, it requires more than 20 (GFLOPs) with an input images (224×224 resolution). Obviously, it keeps us far away to fulfill the demands of running high accuracy and low latency DNN on resource-limited electronic devices. To achieve the goal, there are two approaches have been proposed in the past decades [2]: two steps architecture [3]–[6] and one step [7]–[10] architecture. The one-step architecture DNNs mainly focus on balancing accuracy and latency in opposite to the two-step architecture. Among research efforts in one-step architecture DNN, You Only Look Once (YOLO) architectures [2], [7]–[11] have collected more attention due to its promising performance. Although these efforts significantly improve accuracy and latency, these types of single-shot detector [2], [9]–[12] still requires intensive computation on compute-intensive electronic devices to achieve acceptable accuracy.

There are intensive research efforts on reducing machine learning model's parameters and computations [2], [13]–[15]. Among them, *model compression* techniques have drawn more attention than other methods due to their effectiveness to reduce computational complexity and lessen the memory size [2], [13]–[19]. Although *Tensor Train (TT) decomposition* is a promising mathematical tool to decompose a large tensor to a group of smaller tensor cores, the current TT decomposition for the latest YOLOv5 architecture is still unexplored.

In this paper, we first attempt to develop a TT decomposition method for YOLOv5. Then, we proposed a hardware accelerator design employing software and hardware co-design methodology. Specifically, the main technical contributions are as follows:

- We have developed a novel approach to enable tensor train decomposition to be applied to the YOLOv5 framework. Compared with other compression work, this paper firstly attempts to marriage the TT-decomposition technique with the latest YOLOv5.
- To accelerate the proposed TT-YOLOv5 framework, we propose a novel specialized hardware architecture based on the unique characteristics of tensor train decomposition. The hardware accelerator based on an FPGA device is designated to take advantage of the hardware-friendly TT-YOLOv5 framework we proposed fully and achieves high computation efficiency.

The remainder of this paper is organized as follows: In Section II, we expound the related background. Section III presents the TT-decomposition algorithm for YOLOv5. In Section IV, we introduce the proposed hardware accelerator using software and hardware co-design methodology. Section V shows experimental results using the proposed method to analyze software and hardware performance. Finally, Section VI concludes the manuscript.

II. Related Work

A. Object detectors

In the past decade, target detection using deep learning has been developed rapidly. LeCun et al. [20], [21] firstly proposes a DNN

Research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-20-1-0174. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. This project was also partially supported by National Science Foundation under Grant CCF-1955909.

classification method based on document recognition and extends the ability to detect objects on handwritten images. In general, DNN-based object detectors contain two main categories: (i) two-stage detectors and (ii) one-stage detectors.

Two-stage detectors Two-stage detectors adopted two stages in their architectures: extraction of Region of Interest (RoI) and classification. In the classification stage, classification and bounding box regression are derived out after extracting the RoI. The representative two-stage detectors are R-CNN [4], Fast R-CNN [3], and Faster R-CNN [22]. R-CNN [4] is the first network using a Support Vector Machine (SVM) to perform classification tasks after CNN extracting the features. The depth features extracted by the CNN are used to replace traditional HOG [23], SIFT [24] features, which greatly improve the model efficiency. Some other efforts have been applied to achieve the great improvement both precision and detection performance, such as fast R-CNN and faster R-CNN. For fast R-CNN [3] using softmax to replace SVM for classification, it is noted that the fast R-CNN can efficiently classify object proposals without additional space for storing intermediate features. In particular, faster R-CNN [22] employs a Region Proposal Network (RPN) that stores all detection network features to produce an efficient region proposal. Although these two-stage detectors have achieved the highest accuracy, the main drawback is the high calculation and long inference time according to their working mechanism.

One-stage detectors Using the CNN as a feature extraction network, one-stage detectors regress the position coordinates and category probability of the target object directly without computing the region proposal. Nowadays, SSD [12] [25] , YOLO [7] [8] [9] and RetinaNet [26] have emerged. YOLO can comprehensively regard the whole image, including content information. Consequently, it can decrease the detection error. YOLOv1 [7] employs a standard architecture to extract features from the whole image, and then to predict bounding boxes and categories. To improve speed and precision, YOLOv2 [8], YOLOv3 [9], YOLOv4 [10], and YOLOv5 [11] are proposed. However, the one-stage detector demands GPUs to achieve high-performance computing. Moreover, it takes advantage of an optimized trade-off between accuracy and speed.

Lightweight detectors Lightweight object detectors are proposed for overcoming these issues. SSDlite [27] is proposed to implement SSD on mobile devices by employing an inverted residual structure and lightweight convolutions to extract features. YOLO-lite [28] is a faster and smaller YOLO model based on YOLOv2. YOLObile [2] has proposed an efficient framework on mobile devices through compression-compilation. Co-design based on YOLOv4. Some other works, such as Tiny-DSOD [29] employs a Deeply Supervised Object Detection (DSOD) framework, depth-wise dense block (DDB), and feature-pyramid-network (D-FPN) to adopt resource-constricted devices as its computing devices. However, the accessibility of the Tiny-DSOD to a resource-constricted device leads to a significantly reduced accuracy.

B. You Only Look Once (YOLO) Network v5

You Only Look Once (YOLO) network is an advanced real-time object detection system. As technology evolves, YOLO model keeps iterating to become faster, lighter, and better. Now, the YOLO series contains YOLOv1 [7], YOLOv2 [8], YOLOv3 [9], YOLOv4 [10], and YOLOv5 [11]. As the latest version of YOLO series, YOLOv5 network [11] is developed for a high accuracy and fast inference speed framework. Specifically, it can reach 140 frames per second with a lightweight size and is approximately 90% smaller than YOLOv4, indicating that the YOLOv5 network is very suitable and efficient for deployment to resource-constrained applications real-time object detection tasks. The YOLOv5 framework includes four versions, termed YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x, respectively. The number of model parameters among these four versions increases from YOLOv5s to YOLOv5x. Although YOLOv5s is the smallest among the four versions, the $mAP_{50:95}$ of YOLOv5s under COCO dataset has been reduced to 36.7. Thus, a new design method for more efficient YOLOv5 is needed. The YOLOv5 framework consists of three main pieces, including: *Backbone*, *Neck*, and *Head*, as shown in Fig. 1. In the image prepossessing stage, YOLOv5 load training data with some data augmentation techniques, such as mosaic augmentation, color space adjustment, and scaling. These data augmentation techniques can greatly enrich the dataset and maximize performance. To reduce information redundancy and improve the running speed, the method of auto learning bounding box anchors is introduced to get the best bounding box adaptively during the training processing. Moreover, the images that are scaled and increased by the positive sample anchor can enhance the convergence speed and make the performance of YOLOv5.

Backbone network is a convolutional neural network that aggregates images with contrasting granularities and generates image features. Important modules used in the backbone network include Focus, Conv (or CBL), C3, and SPP. The Focus module is designed to eliminate the computational load of the model and accelerate the operating speed. This module segments the input image with a size of $3 \times 640 \times 640$ into four slices of $3 \times 320 \times 320$ through slicing operation. After that, it uses the concat operation to connect them in depth, which forms the feature map in the size of $12 \times 320 \times 320$. Finally, the output feature map with the size of X $\times 320 \times 320$ is generated through the Conv module that composed of X output channels. C3 is the third module that includes a bottleneck module using the classical residual structure. It is adopted from the CSPNet [30] which is designed to improve depth feature extraction from images. Finally, Spatial Pyramid Pooling (SPP) employs the maximum pooling with a size of 5×5, 9×9 and 13×13 multiscale feature fusion. In all these backbone modules, SPP is embedded to extend the reception range and separate the context by including features with a fixed arbitrary size of the feature vectors.

Neck network is a compounded feature layer consists of a series of blended and mixed image features, which is mainly used for enhancing the detection in applications of objects in multiple scales. YOLOv5 uses PANET(Path Aggregation Network) [31] to aggregate features. To improve the propagation of low-level features, the new FPN architecture is designed by modifying the bottom-up path. Combined with the C3 module, YOLOv5 can greatly enhance the transmission of low-level features and strengthen the network feature fusion ability. Therefore, the network can accurately identify the object with different sizes and scales.

Head network is a prediction network that can produce the output of detection results. Anchor boxes are generated on the output along with a vector that represents the class probability and the bounding box around the object. YOLOv5 contains three detecting layers, which makes the YOLOv5 have the ability to detect targets of different sizes. In addition, GIoU [32] as loss for bounding box regression to further increase the detection accuracy of the algorithm.

III. The Proposed Tensor Train (TT) Decomposition for YOLOv5

Tensor Train (TT) Decomposition is proposed by Oseledets [33] firstly as a spatially efficient method by factorizing higher-dimension tensors into the product of several tensor cores with low-rank.

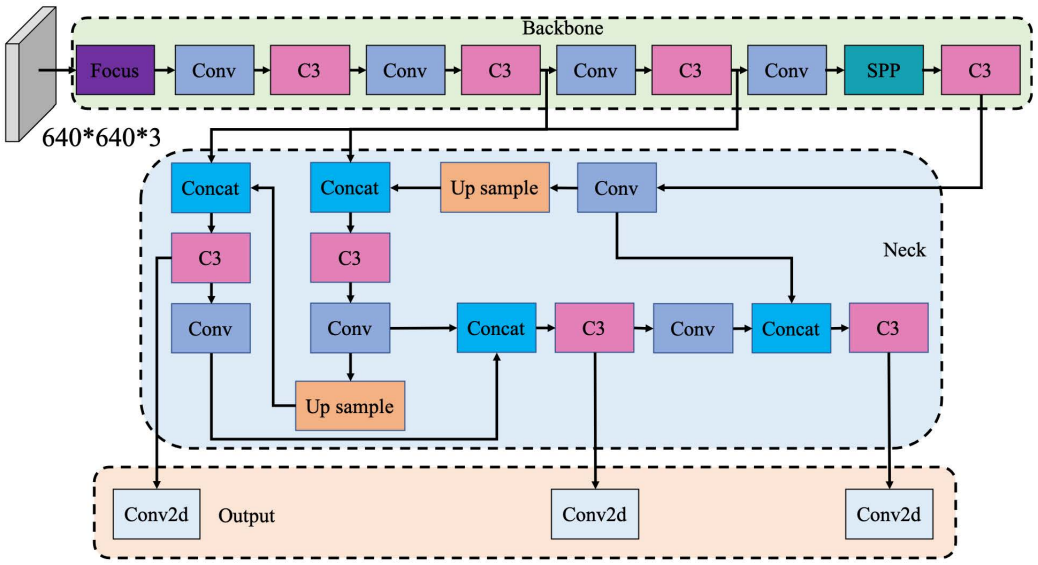

Fig. 1. Architecture of the YOLOv5s model. [11]

Subsequently, Novikov et al. [34] introduce a tensorizing neural network where the TT decomposition is applied to a fully connected layer, which contains very big weight matrices. They tensorize the layer and decompose the weight into several tensor cores using the TT decomposition. Consequently, it can greatly compress the model size and save considerable memory space. Dedicating to a fully shrink weight model, an efficient approach from Garipov et al. [35] greatly reduces the number of parameters by applying TT decomposition to a convolutional layer. Inspired by the previous work, we adopt the TT decomposition to propose a novel framework termed *TT-YOLOv5* to improve the real-time object detection on resource-constricted devices.

A. TT Decomposition

Herein, an N-order tensor can be defined as $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times ... \times I_i \times I_{i+1} \times ... \times I_N}$, $\boldsymbol{\mathcal{A}}(a_1, , ..., a_i, a_{i+1}, ..., a_N)$ is the elements under the tensor $\boldsymbol{\mathcal{A}}$, where $i = 1, 2, ..., N$ and $a_i \in [1, I_i]$. We define the tensor core with $\boldsymbol{\mathcal{G}}_i[a_i] \in \mathbb{R}^{r_{i-1} \times I_i \times r_i}$. Therefore, the N-order tensor can be factorized as:

$$\begin{aligned} &\boldsymbol{\mathcal{A}}(a_1, ..., a_i, a_{i+1}, ..., a_N) = \\ &\boldsymbol{\mathcal{G}}_1[a_1] \cdot ... \cdot \boldsymbol{\mathcal{G}}_i[a_i] \cdot \boldsymbol{\mathcal{G}}_{i+1}[a_{i+1}] \cdot ... \cdot \boldsymbol{\mathcal{G}}_N[a_N] \end{aligned} \tag{1}$$

where the $(r_0, r_1, ..., r_i, r_{i+1}, ..., r_N)$ is defined as the TT-ranks with $r_0 = r_N = 1$, and $\cdot$ denotes the product operation. The complexity of the TT decomposition is determined by the ranks. An N-order tensor $\boldsymbol{\mathcal{A}}$ is represented by the TT-format that requires $\sum_{i=1}^{N} r_{i-1} I_i r_i$ parameters, which is significantly less than the number of original parameters $\prod_{i=1}^{N} I_i$. Please note that the reason why $r_0 = r_N = 1$ is the dimensions multiplied by several tensor cores need to be consistent with the dimensions of original tensor.

B. TT decomposition for vectors and matrices

To make the TT decomposition efficient for large vectors and matrices, we propose a more compact way to express the TT-format for them. Herein, we define boldface letters $\boldsymbol{k}$, $\boldsymbol{K}$ and $\boldsymbol{\mathcal{K}}$ as a vector, a matrix, and a tensor, respectively.

For a vector $\boldsymbol{k} \in \mathbb{R}^Q$, $Q = \prod_{i=1}^{N} q_i$. We can construct a bijective function $F(a) = [f_1(a), ..., f_i(a), f_{i+1}(a), ..., f_N(a)]$ to map the index $a = 1, 2, ..., Q$, where $f_i(a) \in 1, ..., q_i$ and $i \in [1, N]$. Therefore, the TT-format of a vector $\boldsymbol{k}(a)$ can be defined as:

$$\begin{aligned} \boldsymbol{k}(a) =& \boldsymbol{\mathcal{K}}(F(a)) = \boldsymbol{\mathcal{K}}(f_1(a), ..., f_i(a), f_{i+1}(a), ...f_N(a)) \\ =& \boldsymbol{\mathcal{G}}_1[f_1(a)], ..., \boldsymbol{\mathcal{G}}_i[f_i(a)], \boldsymbol{\mathcal{G}}_{i+1}[f_{i+1}(a)], ..., \boldsymbol{\mathcal{G}}_N[f_N(a)] \end{aligned} \tag{2}$$

Consider the matrix $\boldsymbol{K} \in \mathbb{R}^{Q \times P}$, where $Q = \prod_{i=1}^{N} q_i$ and $P = \prod_{i=1}^{N} p_i$, we construct two bijective functions $F(a) = [f_1(a), ..., f_i(a), f_{i+1}(a), ..., f_N(a)]$, $a = 1, 2, ..., Q$ and $G(b) = [g_1(b), ..., g_i(b), g_{i+1}(b), ..., g_N(b)]$, $b = 1, 2, ..., P$ to represent the index (a, b) of the matrix $\boldsymbol{K}$, where $f_i(a) \in (1, ..., p_i), g_i(b) \in (1, ..., q_i), i = 1, 2, ..., N$. Sequentially, $\boldsymbol{K}(a, b)$ as the elements of the $\boldsymbol{K}$ can construct the tensor $\boldsymbol{\mathcal{K}}$ using $F(a)$ and $G(b)$:

$$\begin{aligned} \boldsymbol{K}(a, b) &= \boldsymbol{\mathcal{K}}(F(a), G(b)) \\ &= \boldsymbol{\mathcal{K}}((f_1(a), g_1(b)), ..., (f_i(a), g_i(b)), \\ &\quad (f_{i+1}(a), g_{i+1}(b)), ...(f_N(a), g_N(b))) \\ &= \boldsymbol{\mathcal{G}}_1[(f_1(a), g_1(b)], ..., \boldsymbol{\mathcal{G}}_i[(f_i(a), g_i(b))], \\ &\quad \boldsymbol{\mathcal{G}}_{i+1}[f_{i+1}(a), g_{i+1}(b)], ..., \boldsymbol{\mathcal{G}}_N[f_N(a), g_N(b)] \end{aligned} \tag{3}$$

C. TT convolutional layer for YOLOv5

Due to the complexity and compaction of the YOLOv5 network, we conduct experiments to explore the characteristics of various layers in YOLOv5. Considering the sensitivity of layers in YOLOv5, we develop the TT-YOLOv5 based on our empirical experiments. We use the following formula to express the mode of operation of the convolution kernel:

$$\boldsymbol{Y} = \boldsymbol{X} \otimes \boldsymbol{W} + \boldsymbol{B} \tag{4}$$

$\boldsymbol{X}$ and $\boldsymbol{Y}$ represent the convolution input and output and $\boldsymbol{B}$ represents the bias. To facilitate the convenience of solving the parameters, we rewrite the formula as:

$$\boldsymbol{\mathcal{Y}}_{\dot{h}\dot{w}\dot{c}} = \sum_{m=1}^{k} \sum_{n=1}^{k} \sum_{c=1}^{C} \boldsymbol{\mathcal{W}}_{mnc\dot{c}} \boldsymbol{\mathcal{X}}_{hwc} \tag{5}$$

The conventional 2D convolutional operation takes the 3-dimension input tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{H \times W \times C}$ and outputs a tensor of $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{\dot{H} \times \dot{W} \times \dot{C}}$ using a 4-dimension kernel tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{k \times k \times C \times \dot{C}}$, $m, n =$

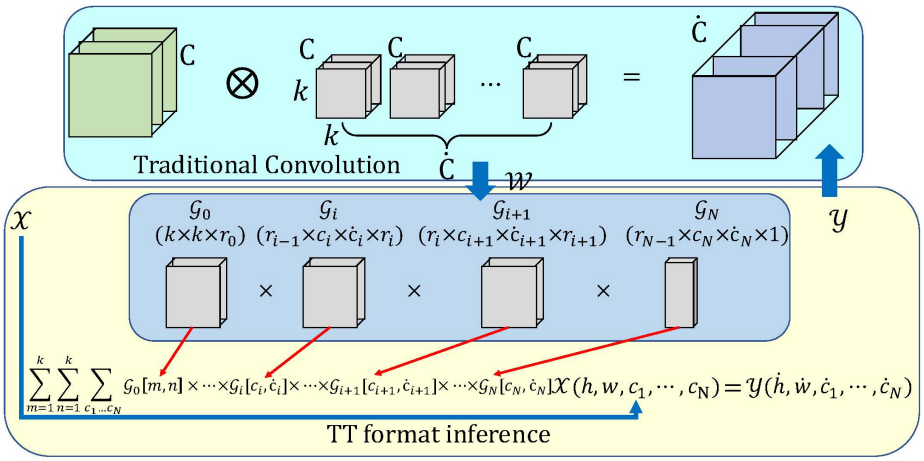

Fig. 2. Tensor train decomposition on convolution layer

$1, 2, ..., k$, and $h \in [1, H]$, $w \in [1, W]$, $\dot{h} \in [1, \dot{H}]$, $\dot{w} \in [1, \dot{W}]$. We denote $\dot{H} = H - m + 1$, $\dot{W} = W - n + 1$. Utilizing the method *Garipov et al.* [35] we have proposed here, the kernel tensor $\boldsymbol{\mathcal{W}}$ can be reshaped as a matrix $\boldsymbol{W}$ with size $k^2 C \times \dot{C}$:

$$\boldsymbol{\mathcal{W}}_{mnc\dot{c}} = \boldsymbol{W}(m + k(n-1) + k^2(c-1), \dot{c}) \tag{6}$$

Then, we reshape the matrix $\boldsymbol{W}$ into the tensor $\dot{\boldsymbol{\mathcal{W}}} \in \mathbb{R}^{k^2 \prod_{i=1}^{N} c_i \dot{c}_i}$, where the dimension $C = \prod_{i=1}^{N} c_i$ and $\dot{C} = \prod_{i=1}^{N} \dot{c}_i$ are decomposed. Thus, this approach makes the TT decomposition more convenient and efficient, because it is flexible to the values of the input channel and output channel by adding some dummy channels filled with zeros. It is noted that N-th dimension has a length of k^2 for $i = 0$ and $C_i \dot{C}_i$ for $i \in [1, N]$.

Therefore, we can obtain the efficient TT decomposition for convolutional kernels given by:

$$\begin{aligned}
&\boldsymbol{W}(m + k(n-1) + k^2(c^* - 1), \dot{c}^*) \\
&= \dot{\boldsymbol{\mathcal{W}}}((m + k(n-1), 1), (c_1, \dot{c}_1), \\
&\quad ..., (c_i, \dot{c}_i), (c_{i+1}, \dot{c}_{i+1}), ..., (c_N, \dot{c}_N)) \\
&= \dot{\mathcal{G}}_0[m + k(n-1), 1] \mathcal{G}_1[c_1, \dot{c}_1] \\
&\quad ... \mathcal{G}_i[c_i, \dot{c}_i] \mathcal{G}_{i+1}[c_{i+1}, \dot{c}_{i+1}] ... \mathcal{G}_N[c_N, \dot{c}_N], \\
&where\ c^* = c_1 + \sum_{i=2}^{N} (c_i - 1) \prod_{j=1}^{i-1} C_j \\
&\quad \dot{c}^* = \dot{c}_1 + \sum_{i=2}^{N} (\dot{c}_i - 1) \prod_{j=1}^{i-1} \dot{C}_j
\end{aligned} \tag{7}$$

To simplify the notation, the formula of the TT decomposition for the convolutional kernel can be expressed as:

$$\begin{aligned}
\dot{\boldsymbol{\mathcal{W}}}_{m,n,c^*,\dot{c}^*} &= \mathcal{G}_0[m, n] \mathcal{G}_1[c_1, \dot{c}_1] ... \mathcal{G}_i[c_i, \dot{c}_i] \\
&\quad \cdot \mathcal{G}_{i+1}[c_{i+1}, \dot{c}_{i+1}] ... \mathcal{G}_N[c_N, \dot{c}_N]
\end{aligned} \tag{8}$$

Based on the theoretical foundation using TT decomposition for YOLOv5, we reconstruct the latest YOLOv5 model using the TT format with the convolutional kernel and propose our model named

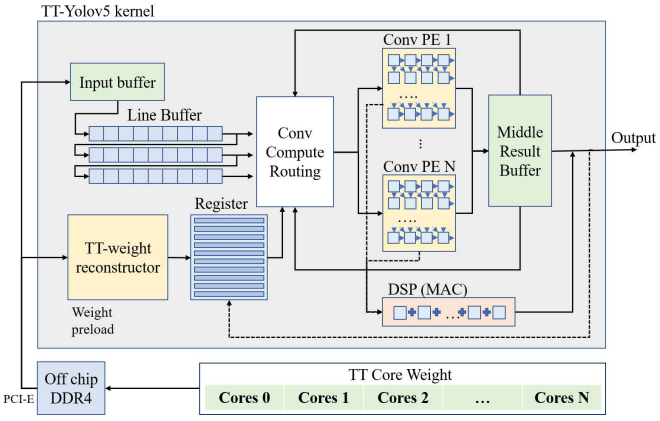

Fig. 3. Overall hardware architecture of TT-YOLOv5 implementation.

TT-YOLOv5. As shown in Fig. 2, convolutional layers in our proposed model can be defined as:

$$\begin{aligned}
\dot{\boldsymbol{y}}_{h,\dot{w},\dot{c}_1,...,\dot{c}_N} &= \sum_{m=1}^{k} \sum_{n=1}^{k} \sum_{c_1,...,c_N} \mathcal{G}_0[m, n] \mathcal{G}_1[c_1, \dot{c}_1] ... \\
&\mathcal{G}_i[c_i, \dot{c}_i] \cdot \mathcal{G}_{i+1}[c_{i+1}, \dot{c}_{i+1}] ... \mathcal{G}_N[c_N, \dot{c}_N] \dot{\boldsymbol{\mathcal{X}}}_{h,w,c_1,...,c_N}
\end{aligned} \tag{9}$$

IV. Overall Architecture of FPGA Accelerator for TT-YOLOv5

Fig. 3 demonstrates the structure of our TT-YOLOv5 framework with its hardware implementation on the FPGA device. Implementing YOLOv5 on resource-constricted devices is still challenging because YOLOv5 is a relatively large model in model size. In this section, we leverage the unique characteristics of the proposed TT decomposition for YOLOv5 to develop a novel hardware architecture. The overall hardware implementation of the proposed TT-YOLOv5 is shown in Fig. 3. The proposed architecture consists of processing elements (PE), off-chip DRAM, MAC cluster, a TT-weight reconstructor, and data buffer. The Processing element is used for implementing the computation. The PE and MAC perform all convolution processes. The TT-weight reconstructor is used to reconstruct the weight for our

compressed tensor cores. Lastly, all data will initially store in DDR4 memory.

In our architecture, the input data and the tensor cores will first be fetched from the off-chip DDR4 memory through PCI-E lanes. Tensor cores will be reconstructed and loaded into the register for computation using the TT-weight reconstructor. The input data will first store in the input buffer and then be broken into one-dimension hardware-friendly data in the line buffer. All data will be read and deployed by the Convolution Compute Routing, and then the routing function sends the control signal to the PEs. Each PE calculates the multiplication and outputs the partial result to the buffer. Depending on the current process, DSP will be used for accumulation, either sending back the result for the next convolution or sending out the final result of each frame.

V. Experiment and Results

In this paper, we propose a novel TT-YOLOv5 framework that performs real-time object detection tasks. Our TT-YOLOv5 framework dedicates to the YOLO framework with resource-restricted devices utilizing the efficient TT decomposition to compress the model size within the high-performance architecture based on FPGA devices. Comparing with state-of-the-art methods, the TT-YOLOv5 framework can achieve remarkable achievement in model size without significant accuracy degradation and throughput increasing. In this section, we implement our proposed work in VOC and COCO datasets. The experimental results both in hardware and software are collected.

A. Experimental Setup

The proposed TT-YOLOv5 framework adopts the TT decomposition with selected convolution layers to keep a balance of model accuracy and size. The training method utilizes PyTorch API, an open-source machine learning library [40]. Our proposed TT-YOLOv5 framework is trained with 300 epochs on Intel i9-9920X + RTX3090 PC, and the hyper-parameters are in the default setup, which is the same as the baseline YOLOv5 model. Specifically, the model's input size is $640 \times 640 \times 3$, and it is trained using MS COCO [41] dataset and PASCAL VOC dataset [42], separately. In addition, we observed that the TT-ranks are vital to the performance of the TT-YOLOv5 framework. In this section, we also explore the variation of various TT-ranks. Therefore, we initialize TT-ranks with various combinations of $[2, 2, 2, 2, 1]$, $[4, 4, 4, 4, 1]$, $[8, 8, 8, 8, 1]$ and $[16, 16, 16, 16, 1]$ and name them as rank 2, rank 4, rank 8 and rank 16, respectively. We perform the experiments on the PyTorch API with the metrics such as mAP and throughput, and the analysis details are given in this section.

B. Comparison to State-of-the-Art in VOC and COCO dataset

Firstly, we evaluate the parameter size and compression ratio of the selected convolution layers, where our TT-decomposition is implemented. We employ several evaluation metrics such as Average Precision under IOU 0.5 (mAP_{50}), the IOU under 0.5 to 0.95 ($mAP_{50:95}$), throughput, and GPU speed. As mentioned above, both accuracy, the number of parameters, and performance are affected by In Table I, we exhibit the impacts of various TT-ranks on our proposed TT-YOLOv5 framework. It is noted that using our proposed TT-YOLOv5 framework can achieve a significant parameter reduction within an accepted accuracy drop. Comparing with the original YOLOv5s model [11], we perform the experiment with the RANK varying from 16 to 2. As result, the size of the selected layers is reduced to 101.4K, 25.9K, 6.73 and 1.81K, which is equivalent to $24\times$, $95\times$, $367\times$, $1363\times$ reduction, respectively. We can also achieve a good performance in terms of other metrics. For instance, the RANK 8 model is trained on the COCO dataset and reaches 34 in $mAP_{50:95}$, 54.1 in mAP_{50}, 17.7 in GFLOPs, and 2.12 ms/img in GPU speed.

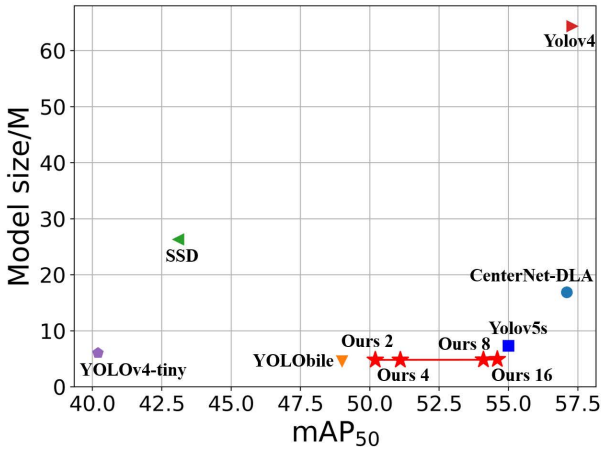

Fig. 4. The model size and mAP comparison in COCO dataset under different models

We further analyze the performance across multiple metrics between the original model and our TT-YOLOv5 model, as shown in Fig. 5. It is noted that the number of parameters in our proposed TT-YOLOv5 is significantly reduced to approximately $2/3$ of the original model, which aligns with the results we present in Table I. Besides the considerable reduction of the number of parameters, the proposed TT-YOLOv5 losses 0.015 mAP_{50} and increase 1.9 GFLOPs throughput, and increase 0.2 GPU speed with the VOC dataset. Similar results are presented with COCO results as well. We also observe that the proposed TT-YOLOv5 varies in performance with different TT-ranks. Smaller TT-ranks can achieve a larger compression ratio but lead to a larger accuracy loss, higher GFLOPs, and GPU speed than other TT-ranks. On the other side, the proposed TT-YOLOv5 provides flexible design freedom to the designers. For example, if the user requires good accuracy, RANK 16 can be selected. However, the trade-off between reduction of model size and accuracy loss and performance drops is not balanced. For example, we can achieve 367 $times$ compression rate with an accuracy loss (0.49 mAP_{50}), a GPU speed increasing (0.02 ms/img), a GFLOPs increasing (0.02).

In Fig. 4, we validate the effectiveness of our TT-YOLOv5 with several representative object detection works. As shown in Fig. 4, it demonstrates the comparison in terms of mAP and model size of TT-YOLOv5 with other reference frameworks. Comparing with the other lightweight object detectors such as YOLOv4-tiny, YOLObile, and YOLOv5s, TT-YOLOv5 has a much higher accuracy but similar model size. On the other hand, TT-YOLOv5 has smallest model size than other detectors such as SSD, CenterNet-DLA, and YOLO4. TT-YOLOv5 has a better accuracy than SSD but worse than CenterNet-DLA and YOLO4.

Finally, we also conduct a comparison with other object detection models using VOC and COCO datasets. Table II reveals that our model makes a good balance between the model performance and the number of parameters. For example, the proposed TT-YOLOv5 with rank-8 in both VOC and COCO dataset achieves a significant parameter drop and throughput improvement with a minor accuracy loss. TT-YOLOv5 with any rank achieves a higher mAP than a full-

TABLE I
PERFORMANCE UNDER DIFFERENT RANKS IN VOC AND COCO DATASET

Ranks	Dataset	$mAP_{50:95}$	mAP_{50}	# Parameters in selected layers(K)	Comp.Rate of selected Conv. Layers	GFLOPs	GPU Speed(ms/img)
Original	VOC 07+12	51.9	78.4	2467.84	1×	16.5	1.7
	COCO [11]	36.7	55.4			17.0	2
16	VOC 07+12	48.8	76.9	101.4	24×	18.4	1.9
	COCO	34.2	54.6			18.9	2.11
8	VOC 07+12	47.5	75.2	25.9	95×	17.2	1.77
	COCO	34	54.1			17.7	2.12
4	VOC 07+12	45.9	73.5	6.73	367×	16.7	1.72
	COCO	31.5	51.1			17.2	2.19
2	VOC 07+12	45.3	73.4	1.81	1363×	16.6	1.71
	COCO	30.6	50.2			17.1	2.33

TABLE II
PERFORMANCE COMPARISON ACROSS MODELS AND DATASETS [2], [36]

Dataset	Models	Input Size	Backbone	$mAP_{50:95}$	mAP_{50}	# Parameters (M)	GFLOPs
VOC 07+12	Faster R-CNN [22]	600	VGG	-	73.2	134.7	-
	Faster R-CNN [22]	600	ResNet-101	-	76.4	-	-
	R-FCN [37]	600	ResNet-101	-	79.5	50.9	-
	SSD300 [12]	300	VGG	-	75.8	26.3	-
	DSSD321 [25]	321	ResNet-101	-	78.6	>52.8	-
	GRP-DSOD320 [36]	320	DS/64-192-48-1	-	78.7	14.2	-
	YOLOv5s	640	-	51.9	78.4	7.11	16.5
	TT-YOLOv5s (Rank 16)	640	-	48.8	76.9	4.74	18.4
	TT-YOLOv5s (Rank 8)	640	-	47.5	75.2	4.67	17.2
	TT-YOLOv5s (Rank 4)	640	-	45.9	73.5	4.65	16.7
	TT-YOLOv5s (Rank 2)	640	-	45.3	73.3	4.64	16.6
COCO	CenterNet-DLA [38]	512	DLA34	39.2	57.1	16.9	52.58
	CornerNet-Squeeze [39]	511	-	34.4	-	31.77	150.15
	SSD [12]	300	VGG16	25.1	43.1	26.29	62.8
	MobileNetv1-SSDLite [27]	300	MobileNetv1	22.2	-	4.31	2.30
	MobileNetv1-SSDLite [27]	300	MobileNetv2	22.1	-	3.38	1.36
	Tiny-DSOD [29]	300	-	23.2	40.4	1.15	1.12
	YOLOV4 [10]	320	CSPDarknet53	38.2	57.3	64.36	35.5
	YOLO-Lite [28]	224	-	12.26	-	0.6	1.0
	YOLOV3-tiny [9]	320	Tiny Darknet	14	29	8.85	3.3
	YOLOV4-tiny [10]	320	Tiny Darknet	-	40.2	6.06	4.11
	YOLObile [2]	320	CSPDarknet53	31.6	49	4.59	3.95
	YOLOv5s [11]	640	-	36.7	55.4	7.3	17.0
	TT-YOLOv5s (Rank 16)	640	-	34.2	54.6	4.90	18.9
	TT-YOLOv5s (Rank 8)	640	-	34	54.1	4.83	17.7
	TT-YOLOv5s (Rank 4)	640	-	31.5	51.1	4.82	17.2
	TT-YOLOv5s (Rank 2)	640	-	30.6	50.2	4.81	17.1

size SSD but lower than CenterNet. However, the TT-YOLOv5 has lower throughput than SSD, YOLOv4, and CenterNet (3.6x, 2.1x and 3x, respectively). We also compare our TT-YOLOv5 with other lightweight detectors such as YOLObile, YOLO-Lite, MobileNetv2-SSDLite, YOLOV4-tiny, and YOLOV3-tiny and note that our TT-YOLOv5 has higher mAP but higher throughput. Compared with the tiny version of YOLOv5s, our TT-YOLOv5 can largely reduce the number of parameters with ignoring accuracy drop and throughput increasing.

C. Hardware Evaluation

Our experimental hardware measurement is based on FPGA evaluation board Xilinx Ultrascale+ KCU116 with the XCKU5P computing unit being connected to a PC through PCI Express. The PCI Express allows the FPGA to access the DDR4 ram as temporary storage for the input data and model. We present the hardware evaluation results in Table III. For TT-YOLOv5, every frame requires a group of convolution computations. The LUT and Flip-Flop (FF) resources are demanded to process the on-chip computation. Off-chip memories are also required for our design because it is not realistic to store the model into the on-chip register only. The LUT collaborates with both register and logic computation modules, and the utilization ratio of this hardware is about 80% from our experiments. The FF is used as the shared register and buffer, while the piped-lining rate is not possible to raise due to the dependency of the data. The FF utilization is 20-30% and varies with the model compressing ratio. The BRAM utility remains the same because of using tensor shape manipulation that is designated to hold more space for redundancy and not tightly allocate the data and model. The BRAM utility in VOC 07+12 dataset is 45.8% and 49% in the COCO dataset. To process the convolution, we design MAC (Multiply Accumulate) as processing units that consist of multiple DSPs. We observe that the higher the parallelism channel lane deployed, the higher DSP utilization will be obtained. Thus, our utilization rate reaches about 65% to 75% of the available resources. To raise the speed of the process, we can invest more DSP resources for the design. Table III, we measure the power consumption within two places: on-chip and off-chip. The on-chip power is drained by all on-chip resources and

TABLE III
HARDWARE EVALUATION IN DIFFERENT DATASETS AND RANKS COMPARISON.

Ranks	Dataset	Model Size(MB)	Param (M)	LUT	Util (%)	FF	Util (%)	BRAM	Util (%)	DSP	Util (%)	DDR4 Power (W)	Onchip Power (W)	GOPS
16	VOC 07+12	9.76	4.74	182,022	83.9	123,098	28.4	220	45.8	1,321	72.4	1.30	15.3	42.6
8	VOC 07+12	9.61	4.67	175,330	80.8	122,088	28.1	220	45.8	1,321	72.4	1.31	14.7	43.8
4	VOC 07+12	9.57	4.65	174,341	80.4	85,230	19.6	220	45.8	1,212	66.4	1.28	12.3	44.1
2	VOC 07+12	9.56	4.64	168,050	77.5	85,002	19.6	220	45.8	1,200	65.8	1.28	12.1	44.9
16	COCO	10.1	4.9	187,022	86.2	143,728	33.1	235	49	1,351	74.1	1.33	16.1	34.2
8	COCO	9.93	4.83	180,330	83.1	132,018	30.4	235	49	1,335	73.2	1.33	15.7	34.5
4	COCO	9.89	4.82	177,541	81.8	126,642	29.2	235	49	1,312	71.9	1.31	14.3	36.2
2	COCO	9.88	4.81	170,050	78.4	120,132	27.7	235	49	1,280	70.2	1.28	14.1	36.4

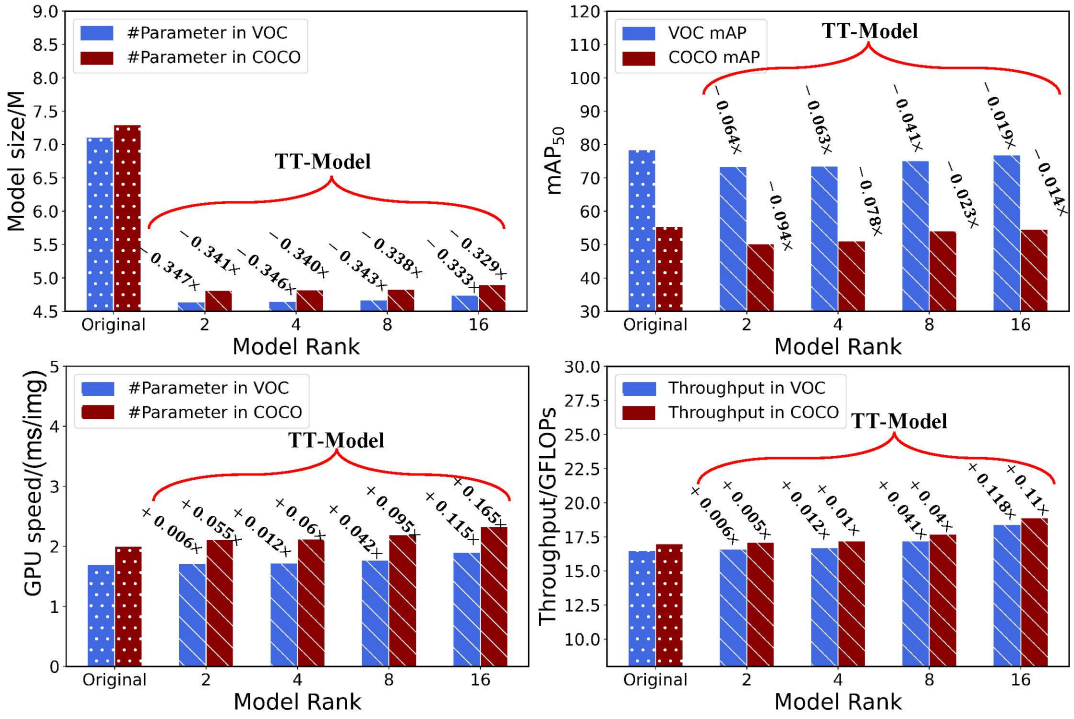

Fig. 5. GPU speed, throughput, and mAP_{50} comparison of our TT-YOLOv5 and original YOLOv5s.

the connection between the chip and the I/O supply by the evaluation board. Off-chip power is consumed by the DDR4 memory that is connected through the PCI-E to the FPGA. Table III presents the overall power of the VOC dataset. TT-YOLOv5, with rank 16, reaches the highest power usage (16.6 W). TT-YOLOv5 with rank 2 reaches the smallest power consumption (13.38 W). On the opposite way, TT-YOLOv5, with rank 16, presents the smallest GOPS (42.6). In the COCO dataset, the highest power is consumed by TT-YOLOv5 with rank 16 (17.43 W), and the smallest power is consumed by TT-YOLOv5 with rank 2 (15.38 W). The GOPS of the various TT-ranks in both datasets exhibits the highest value in the rank 2 models (44.9 in the VOC dataset and 36.4 in the COCO dataset).

VI. CONCLUSION

This paper presents a novel Tensor Train (TT) based YOLOv5 framework and its hardware implementation on the FPGA board. Comparing with the original tiny version of the YOLOv5 model (YOLOv5s), our TT-YOLOv5 achieves a significant reduction in parameter size with less than 0.094× accuracy drop in mAP_{50}, 0.165× GPU speed increasing, and 0.118× increase in throughput. Specifically, we reach 24×, 95×, 367×, 1363× in compression ratio with various RANKs 16, 8, 4, 2 respectively. Moreover, our TT-YOLOv5 can achieve a balance of performance and model size compared to other state-of–of-art object detection models. For the TT-YOLOv5 with RANK 8 model, we observe 47.5 and 34 in $mAP_{50:95}$, 75.2 and 54.1 in mAP_{50}, 4.67 and 4.83 in parameter size, 17.2 and 17.7 GFLOPs in throughput under VOC and COCO dataset respectively. Our hardware measurements show that the proposed TT-YOLOv5 can perform an efficient computation in both VOC and COCO datasets with small power consumption and hardware usage. Thus, our TT-YOLOv5 and its hardware design can benefit the research of edge computation and its applications.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[2] Y. Cai, H. Li, G. Yuan, W. Niu, Y. Li, X. Tang, B. Ren, and Y. Wang, "Yolobile: Real-time object detection on mobile devices via compression-compilation co-design," *arXiv preprint arXiv:2009.05697*, 2020.

[3] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[5] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 925–938.

[6] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[8] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[9] ——, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[10] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[11] G. Jocher, A. Stoken, J. Borovec, NanoCode012, ChristopherSTAN, L. Changyu, Laughing, tkianai, yxNONG, A. Hogan, and et al., "ultralytics/yolov5: v4.0 - nn.silu() activations, weights & biases logging, pytorch hub integration," Jan 2021.

[12] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[13] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *arXiv preprint arXiv:1608.03665*, 2016.

[14] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," *arXiv preprint arXiv:1608.04493*, 2016.

[15] C. Min, A. Wang, Y. Chen, W. Xu, and X. Chen, "2pfpce: Two-phase filter pruning based on conditional entropy," *arXiv preprint arXiv:1809.02220*, 2018.

[16] Y. Yang *et al.*, "Tensor-train recurrent neural networks for video classification," in *ICML*, 2017, pp. 3891–3900.

[17] J. Ye *et al.*, "Learning compact recurrent neural networks with block-term tensor decomposition," in *CVPR*, 2018, pp. 9378–9387.

[18] Y. Pan *et al.*, "Compressing recurrent neural networks with tensor ring for action recognition," in *AAAI*, vol. 33, 2019, pp. 4683–4690.

[19] M. Yin, S. Liao, X.-Y. Liu, X. Wang, and B. Yuan, "Compressing recurrent neural networks using hierarchical tucker tensor decomposition," *arXiv preprint arXiv:2005.04366*, 2020.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[22] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[23] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. Ieee, 2005, pp. 886–893.

[24] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International journal of computer vision*, vol. 128, no. 2, pp. 261–318, 2020.

[25] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "Dssd: Deconvolutional single shot detector," *arXiv preprint arXiv:1701.06659*, 2017.

[26] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[28] R. Huang, J. Pedoeem, and C. Chen, "Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 2503–2510.

[29] Y. Li, J. Li, W. Lin, and J. Li, "Tiny-dsod: Lightweight object detection for resource-restricted usages," *arXiv preprint arXiv:1807.11013*, 2018.

[30] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 2020, pp. 390–391.

[31] J. Yang, X. Fu, Y. Hu, Y. Huang, X. Ding, and J. Paisley, "Pannet: A deep network architecture for pan-sharpening," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5449–5457.

[32] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 658–666.

[33] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[34] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov, "Tensorizing neural networks," *arXiv preprint arXiv:1509.06569*, 2015.

[35] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, "Ultimate tensorization: compressing convolutional and fc layers alike," *arXiv preprint arXiv:1611.03214*, 2016.

[36] Z. Shen, H. Shi, R. Feris, L. Cao, S. Yan, D. Liu, X. Wang, X. Xue, and T. S. Huang, "Learning object detectors from scratch with gated recurrent feature pyramids," *arXiv preprint arXiv:1712.00886*, vol. 1, 2017.

[37] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," *arXiv preprint arXiv:1605.06409*, 2016.

[38] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 6569–6578.

[39] H. Law, Y. Teng, O. Russakovsky, and J. Deng, "Cornernet-lite: Efficient keypoint based object detection," *arXiv preprint arXiv:1904.08900*, 2019.

[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[41] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[42] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.