Article

Autonomous Binarized Focal Loss Enhanced Model Compression Design Using Tensor Train Decomposition

Mingshuo Liu [1], Shiyi Luo [1], Kevin Han [1], Ronald F. DeMara [2,*] and Yu Bai [1,*]

[1] Electrical and Computer Engineering Department, College of Engineering and Computer Science, California State University, 800 N State College Blvd, Fullerton, CA 92831, USA
[2] Department of Electrical and Computer Engineering, College of Engineering and Computer Science, University of Central Florida, 4000 Central Florida Blvd, Orlando, FL 32816, USA
* Correspondence: Ronald.Demara@ucf.edu (R.F.D.); ybai@fullerton.edu (Y.B.); Tel.: +1-407-823-5916 (R.F.D.); +1-657-278-5359 (Y.B.)

Abstract: Deep learning methods have exhibited the great capacity to process object detection tasks, offering a practical and viable approach in many applications. When researchers have advanced deep learning models to improve their performance, the model derived from the algorithmic improvement may itself require complementary increases in computational and power demands. Recently, model compression and pruning techniques have received more attention to promote the wide employment of the DNN model. Although these techniques have achieved a remarkable performance, the class imbalance issue during the mode compression process does not vanish. This paper exploits the Autonomous Binarized Focal Loss Enhanced Model Compression (ABFLMC) model to address the issue. Additionally, our proposed ABFLMC can automatically receive the dynamic difficulty term during the training process to improve performance and reduce complexity. A novel hardware architecture is proposed to accelerate inference. Our experimental results show that the ABFLMC can achieve higher accuracy, faster speed, and smaller model size.

Keywords: tensor decomposition; focal loss; embedded hardware

Citation: Liu, M.; Luo, S.; Han, K.; DeMara, R.F.; Bai, Y. Autonomous Binarized Focal Loss Enhanced Model Compression Design Using Tensor Train Decomposition. *Micromachines* **2022**, *13*, 1738. https://doi.org/10.3390/mi13101738

Academic Editor: Arman Roohi

Received: 1 September 2022
Accepted: 11 October 2022
Published: 14 October 2022

1. Introduction

The current state-of-the-art object detection network using Deep Learning conducts a competition between various models due to their incredible feats in the field. Deep learning techniques are widely adopted in various applications, such as self-driving UAVs, Water Quality Prediction [1], autonomous robotics, and robot vision. It is clearly seen that these tasks demand a Deep Learning technology with high accuracy, smaller size, and low latency model running on mobile electronic devices.

Among various deep learning techniques in the object detection field based on Convolutional Neural Networks (CNNs), there are two approaches that can be summarized in the past decades [2]: one stage approach [3] and two-stage approach [4]. Although the latter method achieves a remarkable accuracy performance on object detection benchmarks COCO [5], the models suffer from a longer execution time. In contrast, a one-stage detector runs faster, suffering from poorer accuracy. Although these one- and two-stage approaches have demonstrated powerful capabilities in many applications, the previous research paper [6] has indicated that class imbalance has been identified as a key effect of this performance gap. Two-stage approaches can immune this imbalance problem as they remove the background before identifying the objects [7]. In addition, these approaches use fixed background ratio [4] or online hardware example mining [8] to balance the training data in the second stage. In contrast, the one-stage detector faces more challenges since it needs to learn the difference between foreground classes and possible scenery. Thus, the two approaches mentioned above cannot be applied directly to a one-stage detector.

Furthermore, although class imbalance has been addressed in the R-CNN series detectors using a two-stage cascade and sampling heuristics [6], the one-stage detector must process large candidate locations in an image. For example, in practice, the detector may receive $\sim$100K candidate locations in the image. Thus, it is inefficient during the training procedure when the samples are dominated by easily classified background examples.

Recent work has been proposed to employ α balancing to the loss function to distinguish classes and backgrounds. This method is intended to improve the detector to avoid imbalance classification. After this first attempt toward a dynamic loss function, RetinaNet [6] has added a data dynamic to the loss function, named Focal Loss. The probability-based difficulty for the correct class is calculated differently in each classification. In this work, a hyper-parameter γ is added to control the loss function. However, different neural network models may require different hyper-parameters to obtain the best accuracy. In our earlier work [9,10], we have shown that the class imbalance issue gets worse during model compression, and we have demonstrated that the Focal Loss can improve the model accuracy within imbalanced data. However, the proposed Focal Loss is difficult to obtain preliminary information and fine-tune the parameters.

Limitations of Existing Imbalance Issue on Model Compression: Although the model compression approach plays an important role in developing efficient deep leanings [11,12], the state-of-the-art model compression methods still suffer from a class imbalance issue due to three challenging limitations. First, the existing works only focus on the model size and accuracy of the model as its target objectives without considering the issue of class imbalance. Second, although TT-decomposition achieves a remarkable result in compressing the model without significant accuracy drops, the added more tensor cores create a complex parameter setting when we apply focal loss technology directly to the model compression algorithm. Third, current focal loss methods have been implemented only on software platforms. Thus, such focal loss methods can only achieve suboptimization and lead to performance drop on model compression when we employ the resource constraint hardware.

In this paper, a novel approach constructing an autonomous focal loss algorithm that performs an efficient loss function for class imbalance issues during model compression is presented. We first attempt to address the imbalance issue during the model compression. In addition, a high-performance hardware accelerator is developed in this paper. Specifically, the main technical contributions are as follows:

- In this work, we first attempt to consider the class imbalance issue during compression of the tensor train-based model. The proposed ABFLMC algorithm can be used for the tensor train decomposition method to overcome the class imbalance issue.
- At the algorithm level, the proposed ABFLMC algorithm is designated by considering the characteristics of the tensor train decomposition to reduce the complexity and increase the performance. Consequently, the proposed framework can automatically search for the best parameters for overcoming the imbalance issue during the training process.
- At the hardware design level, the architecture of the proposed ABFLMC algorithm has been developed to maximize parallelism and processing throughput. Thus, the proposed ABFLMC algorithm can achieve an optimized solution on the resource constraint hardware.

The remainder of this paper is organized as follows: In Section 2, we introduce the related background in object detection, focal Loss, and tensor train decomposition. In Section 3, we demonstrate a novel algorithm for optimizing autonomous focal loss models (ABFLMC) to overcome unbalanced issues during model compression. Section 4 discusses the hardware design of the ABFLMC model. Section 5 presents experimental results in both software and hardware. Finally, Section 6 concludes the manuscript.

2. Related Work

2.1. Object Detection

Object detection is a fundamental task in the field of computer vision. The purpose of an object detector is to classify and localize all objects in an image or video. Object detectors are designed to extract hand-crafted features [13,14], which are widely employed in various branches like self-driving cars. With the increasing size of datasets, object detectors become larger and larger, leading to decreased speed and accuracy. Thus, much research has been done to improve object detectors. Generally, object detectors can be summarized into two categories according to the structure: one-stage and two-stage detectors.

Two-Stage Detectors: In paper [4], Girshick et al. propose a region-based convolutional neural network (R-CNN) using AlexNet [15] as the backbone. This framework employs the region proposal module to extract features. After the feature is extracted, it is classified by class-specific support vector machines (SVMs) to obtain scores. Although it achieves good performance, the longer execution time to detect an image is a drawback. Some research efforts such as Fast R-CNN [16] have been proposed to overcome this problem. The Fast R-CNN is trained with an end-to-end architecture, and a multitask loss, which is very simple for the training and can improve operation speed and forecast accuracy. After researchers constantly strive for excellence, a novel model that combines Fast R-CNN and a region proposal network (RPN) is proposed [7]. This faster R-CNN can learn and generate better region proposals using CNN used in the region proposal module, leading to improved accuracy. Later, an R-FCN [17] combined with faster R-CNN and FCN is presented to reduce model training time. The proposed R-FCN can improve the 2.5–20$\times$ speed compared to Faster R-CNN. However, since it uses ResNet101, the model is difficult to implement on devices with limited resources. Recently, DetectoRS [18] employing Recursive Feature Pyramid (RFP), Atrous Spatial Pyramid Pooling (ASPP), and Switchable Atrous Convolution (SAC) is proposed. This work builds a switch system to control the rate of convolution. Consequently, it can improve the detection of multi-scale objects. Although the DetectoRS have improved the model performance, it is still ineffective in real-time tasks due to its complexity.

One-Stage Detectors: Compared with two-stage detectors, the one-stage detectors exchange object detector tasks from classification to a regression. You Only Look Once (YOLO) [19] utilizes multiple smaller convolutional networks in a cascading way to predict the image directly with a bounding box. An image is divided into $N \times N$ parts, and each of the parts must predict multiple bounding boxes. In this way, the YOLO model achieves a significant improvement in both running speed and accuracy. Following the YOLO Model, Single Shot MultiBox Detector (SSD) [3] is proposed. It can balance the speed and accuracy of real-time detection tasks. Its accuracy achieves similar performance to a two-stage detector Faster R-CNN. Later, a variant of the YOLO network-YOLOv2 (YOLO9000) [20] is presented. It uses Darknet-19 [21] as the backbone architecture and uses multiple efficient techniques such as Batch Normalization [22], WordTree [23] to improve efficiency. The fully-connected layer is removed to enhance the inference speed. YOLOv2 becomes "better, faster, and stronger". Within the YOLO series, Yolov4 [24] is proposed for object detection and achieves a new record speed (65FPS).

Many innovations are implemented within the framework of YOLOv4. Specifically, the "Bag of Freebies" includes class label smoothing, data augmentation, and the Cross mini-Batch Normalization (CmBN) algorithm that collects the statistics between the mini-batches, these technologies are efficient to improve model performance without increasing the inference time. However, the "Bag of Special" with CSP (Cross-stage partial connection), SPP module, and PAN neck still leads to the inference time rise [25]. The latest variant of YOLO YOLOv5 [26] has been proposed to improve YOLOV4. However, in general, these object detectors are not feasible on resource-constrained devices such as non-GPU laptops and mobile devices. Current research efforts, including MobileNet-SSDLite [27], develop lightweight depth-wise convolutions to extract features and use the delinearized module in low-dimensional layers. Both the execution time of the operation and the size of the model are decreased while maintaining the same accuracy. Following this work, YOLO-LITE [28] and Tiny-DSOD [29] aim to create a faster, smaller, and high-performed model enhancing the accessibility that makes real-time detection model deployed on all devices. Moreover, many compression algorithms like pruning and tensor-train decomposition are introduced to improve computational efficiency. YOLObile [30] presents a novel block-punched pruning scheme that exhibits a high accuracy on mobiles and embedded devices.

Class Imbalance Issue: In the paper [6], Tsung-Yi Lin et al. state that the object detection models encounter the class imbalance problem while training. There are 10^4–10^5 potential samples per image; however, a small amount of them comprise the desired objects. The imbalance would make training inefficient and disturb the training process because of many easy negative samples. To address this issue, they came up with the focal loss. This focal loss can naturally deal with class imbalance without the traditional sampling method by applying the balance factor and modulating the factor into the loss function.

2.2. Tensor Decomposition Methods

The tensors are important for an efficient computation since a large number of dimensions of tensors demands an intensive memory. The number increases exponentially with the number of dimensions. Over the last decades, Many notable tensor decomposition approaches are proposed, including Tucker decomposition [31], the canonical polyadic decomposition (CPD) [32–34], parallel factor (PARAFAC2) [35], CTSVD-QR [36] and tensor train decomposition (TT decomposition) [10,37,38]. Among these approaches, TT decomposition is an impressive method with some benefits, such as efficient tensor representation (TT format) dedicated to reducing memory storage and flexible and high-efficiency reasoning logic with singular value decomposition. Moreover, experimental results in [39] show that TT decomposition can be applied to fully connected (FC) layers, resulting in a significant reduction in the number of parameters with a tiny drop in accuracy. After that, Novikov1 et al. adopt the TT decomposition to tensorflow for its easier utilization [40]. Subsequently, Garipov et al. [41] propose a novel method that applies TT decomposition to both convolutional layers and fully-connected layers. Therefore, the TT decomposition approach has become a very promising model compression tool.

In detail, TT decomposition is capable of factorizing a d-dimensional tensor with the size of $n_1 \times n_2 \times \ldots \times n_d$ into several tensor cores with the size of $r_{k-1} \times n_k \times r_k$. To expound the concept of TT decomposition, we utilize the naming convention from Garipov et al. [41]. A tensor is defined as $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$. The TT-representation of the tensor $\boldsymbol{\mathcal{A}}$ is a set of TT-cores $\boldsymbol{G}_k[j_k] \in \mathbb{R}^{r_{k-1} \times r_k}$, where $j_k \in [1, n_k], k = 1, 2, \ldots, d$. Therefore, $\boldsymbol{\mathcal{A}}(j_1, j_2, \ldots, j_d)$ as the element of the tensor $\boldsymbol{\mathcal{A}}$ can be decomposed as:

$$\boldsymbol{\mathcal{A}}(j_1, j_2, \ldots, j_d) = \boldsymbol{G}_1[j_1]\boldsymbol{G}_2[j_2]\ldots\boldsymbol{G}_d[j_d] \tag{1}$$

where r_k is the rank value, and $r_0 = r_d = 1$ are adopted to ensure the feasibility of TT decomposition. After the decomposition of the TT, the number of parameters of the tensor $\boldsymbol{\mathcal{A}}$ is $\sum_{k=1}^{d} r_{k-1}n_k r_k$, which is significantly less than the original size $\prod_{k=1}^{d} n_k$. Therefore, the compression ratio can be defined as $\prod_{k=1}^{d} n_k / \sum_{k=1}^{d} r_{k-1}n_k r_k$, and choosing the balanced value of r_k is significant in maximizing the compression ratio.

3. Autonomous Binarized Focal Loss Enhanced Model Compression Algorithm (ABFLMC)

3.1. TT-Convolutional Layer in the Model YOLOV5

Garipov et al. [41] point out that using the TT-decomposition to factorize the convolutional kernel into the product of several low-rank matrices directly has the limitation on the convolutional layer. To address this drawback, a new decomposition is introduced that can be applied to the convolutional and fully connected layers. For a convolutional layer, assume that the input tensor $\boldsymbol{\mathcal{I}}$ has the size $\boldsymbol{W}_I \times \boldsymbol{H}_I \times \boldsymbol{C}_I$, the output tensor $\boldsymbol{\mathcal{O}} \in \mathbb{R}^{\boldsymbol{W}_O \times \boldsymbol{H}_O \times \boldsymbol{C}_O}$, and the kernel tensor $\boldsymbol{\mathcal{T}}$ is a 4-dimensional tensor with the size $K \times K \times \boldsymbol{C}_I \times \boldsymbol{C}_O$. Now, we can easily write the formula for the operation of a YOLOV5 convolutional layer as: $\mathcal{O}(w,h,c_O) = \sum_{i=1}^{K}\sum_{j=1}^{K}\sum_{c_I=1}^{C_I}\mathcal{T}(i,j,c_I,c_O)\mathcal{I}(w+i-1,h+j-1,c_I)$. Figure 1 shows the Tensor Train process and the element shift process. The convolutional formula can also be calculated as several matrix multiplications $\boldsymbol{O} = \boldsymbol{I} \times \boldsymbol{T}$. The top in Figure 1 shows how the input tensor is decomposed into several matrix multiplications. The purpose of the following derivation is to express the relationship between the input tensor, output tensor, and the kernel tensor via introducing how the n-th row of the input matrix is used to compute the n-th row of the output of the matrix multiplication. First, we transfer the input tensor $\mathcal{I} \in \mathbb{R}^{\boldsymbol{W_I}\times\boldsymbol{H_I}\times\boldsymbol{C_I}}$ and the output tensor $\mathcal{O} \in \mathbb{R}^{\boldsymbol{H_O}\times\boldsymbol{W_O}\times\boldsymbol{C_O}}$ to matrices. To analyze the process, we define a patch of the input tensor within size $K \times K \times \boldsymbol{C_I}$ and a patch of the output tensor is $1 \times 1 \times \boldsymbol{C_O}$, so we easily obtain the connection $\boldsymbol{H_O} = \boldsymbol{H_I} - K + 1$ and $\boldsymbol{W_O} = \boldsymbol{W_I} - K + 1$. Now, we remodel the output tensor $\mathcal{O}$ into a matrix $\boldsymbol{O}$: $\mathcal{O}(w,h,c_O) = \boldsymbol{O}(w + \boldsymbol{W_O}(h-1), c_O)$, where $w + \boldsymbol{W_O}(h-1)$ represents the position of the patch in the plane $\boldsymbol{H_O} \times \boldsymbol{W_O}$ of the output tensor that is transferred to the height of the output matrix $\boldsymbol{O}$. Furthermore, c_O is the width of matrix $\boldsymbol{O}$, where $w \in (1,\ldots,\boldsymbol{W_O})$ and $h \in (1,\ldots,\boldsymbol{H_O})$. In this way, the 3-dimensional tensor can be reshaped into a 2-dimensional matrix. Therefore, the input tensor $\mathcal{I}$ also can be resized as follows:

$$\begin{aligned}&\mathcal{I}(w+i-1,h+j-1,c_I) = \\ &\boldsymbol{I}(w+\boldsymbol{W_O}(h-1), i+K(j-1)+K^2(c_I-1)) \\ &where\ i,j \in (1,\ldots,K)\end{aligned} \tag{2}$$

Then, we can obtain the kernel matrix of size $K^2\boldsymbol{C_I} \times \boldsymbol{C_O}$ from the kernel tensor $\mathcal{T}$: $\mathcal{T}(i,j,c_I,c_O) = \boldsymbol{T}(i+K(j-1)+K^2(c_I-1), c_O)$.

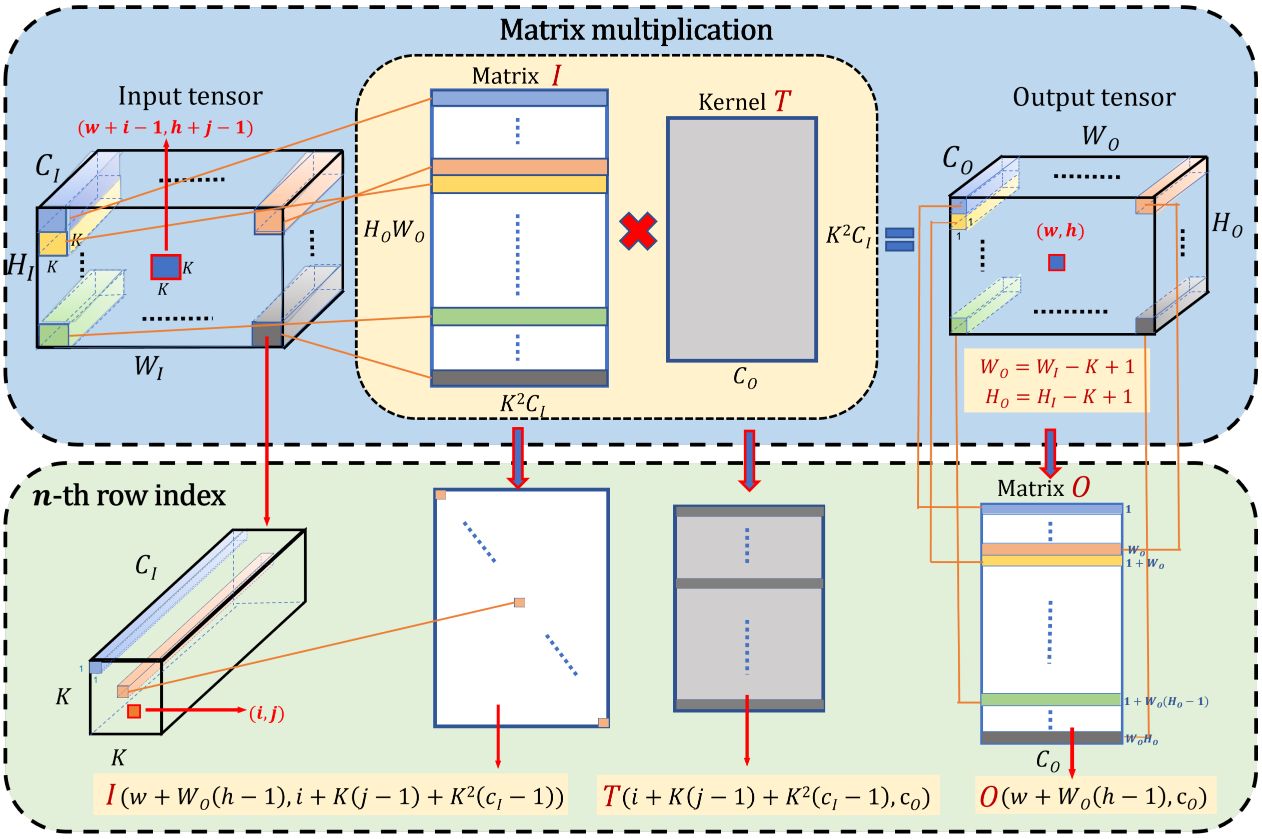

Figure 1. Tensor Train convolutional process: The top shows a convolutional layer can be reformulate to a matrix-by-matrix multiplication $\boldsymbol{O} = \boldsymbol{I} \times \boldsymbol{T}$, the bottom introduces how to calculate the n-th row of the matrices that correspond to the $K \times K \times C_I$ patch of the input tensor as Equation (2) illustrates.

In the following work, the TT format is introduced to decompose the matrix using the coincidence of the TT decomposition and the low-rank decomposition. More specifically, we have reshaped the tensor into the matrix before, then we transfer the matrix to a more compact tensor, and TT-decomposition is applied on the new tensor to obtain the matrix format TT. Let us assume a matrix $\boldsymbol{X} \in \mathbb{R}^{M \times N}$, $M = \prod_{a=1}^{d} m_a$ and $N = \prod_{a=1}^{d} n_a$. Two objective functions can be constructed to shape the matrix $\boldsymbol{X}$ into the tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 m_1 \times n_2 m_2 \times \ldots \times n_a m_a}$ in the following way:

$$\begin{aligned} &F(i) = [f_1(i), \ldots f_a(i), \ldots, f_d(i)] \\ &G(j) = [g_1(j), \ldots g_a(j), \ldots, g_d(j)] \\ &\textit{where } f_a(i) \in (1, \ldots, m_a) \;\; g_a(j) \in (1, \ldots, n_a) \\ &\qquad a \in (1, \ldots, d) \; i \in (1, \ldots, M) \; j \in (1, \ldots, N) \end{aligned} \tag{3}$$

Sequentially, using the TT-format to represent the elements, the $\boldsymbol{X}(i, j)$ is defined as:

$$\begin{aligned} &X(i,j) \\ &= \boldsymbol{\mathcal{X}}((f_1(i), g_1(j)), \ldots, (f_a(i), g_a(j)), \ldots, (f_d(i), g_d(j))) \\ &= \boldsymbol{G}_1[(f_1(i), g_1(j)] \ldots \boldsymbol{G}_a[(f_a(i), g_a(j)] \ldots \boldsymbol{G}_d[(f_d(i), g_d(j)] \end{aligned} \tag{4}$$

According to the TT-representation, it can reshape these matrices $\boldsymbol{I}, \boldsymbol{O}$, and $\boldsymbol{T}$ into new tensors $\hat{\boldsymbol{\mathcal{I}}}, \hat{\boldsymbol{\mathcal{O}}}$, and $\hat{\boldsymbol{\mathcal{T}}}$. Herein, we firstly define $C_I = \prod_{a=1}^{d} C_{I_a}$ and $C_O = \prod_{a=1}^{d} C_{O_a}$, the output matrix $\boldsymbol{O}$ can be reshaped into a new tensor $\hat{\boldsymbol{\mathcal{O}}}$ with size $\boldsymbol{W}_O \times \boldsymbol{H}_O \times \boldsymbol{C}_{O_1} \times \ldots \times \boldsymbol{C}_{O_a} \ldots \times \boldsymbol{C}_{O_d}$, then the new input tensor is $\hat{\boldsymbol{\mathcal{I}}}$ of size $(\boldsymbol{W}_I + K - 1) \times (\boldsymbol{H}_I + K - 1) \times \boldsymbol{C}_{I_1} \times \ldots \times \boldsymbol{C}_{I_a} \ldots \times \boldsymbol{C}_{I_d}$. For the kernel tensor, we deduce the formula according to Equation (4) as:

$$\begin{aligned} &\boldsymbol{T}(i + K(j-1) + K^2(\hat{c_I} - 1), \hat{c_O}) \\ &= \hat{\boldsymbol{\mathcal{T}}}((i + K(j-1), 1), (c_{I_1}, c_{O_1}), \ldots, (c_{I_a}, c_{O_a}), \ldots, (c_{I_d}, c_{O_d})) \\ &\quad \textit{where } \hat{c_I} = c_{I_1} + \sum_{i=2}^{d} (c_{I_i} - 1) \prod_{j=1}^{i-1} c_{I_j} \\ &\qquad\quad \hat{c_O} = c_{O_1} + \sum_{i=2}^{d} (c_{O_i} - 1) \prod_{j=1}^{i-1} c_{O_j} \end{aligned} \tag{5}$$

Then we use TT-decomposition to factorize the kernel tensor as:

$$\begin{aligned} &\boldsymbol{T}(i + K(j-1) + K^2(\hat{c_I} - 1), \hat{c_O}) \\ &= \hat{\boldsymbol{G}}_0[i + K(j-1), 1] \boldsymbol{G}_1[c_{I_1}, c_{O_1}] \ldots \boldsymbol{G}_a[c_{I_a}, c_{O_a}] \ldots \boldsymbol{G}_d[c_{I_d}, c_{O_d}] \end{aligned} \tag{6}$$

where $\hat{\boldsymbol{G}}_0$ is the tensor core related to the convolution kernel and $\boldsymbol{G}_1$ to $\boldsymbol{G}_d$ are the regular tensor cores as we mentioned above. Finally, the convolution layer can be rewritten using the TT-format:

$$\begin{aligned} &\hat{\boldsymbol{\mathcal{O}}}(w, h, c_{O_1}, \ldots, c_{O_a}, \ldots, c_{O_d}) \\ &= \sum_{i=1}^{K} \sum_{j=1}^{K} \sum_{c_{I_1}, \ldots, c_{I_a}, \ldots, c_{I_d}} \hat{\boldsymbol{\mathcal{I}}}(w + i - 1, h + j - 1, c_{I_1}, \ldots, c_{I_a}, \ldots, c_{I_d}) \\ &\hat{\boldsymbol{G}}_0[i + K(j-1), 1] \boldsymbol{G}_1[c_{I_1}, c_{O_1}] \ldots \boldsymbol{G}_a[c_{I_a}, c_{O_a}] \ldots \boldsymbol{G}_d[c_{I_d}, c_{O_d}] \end{aligned} \tag{7}$$

3.2. Design of ABFLMC

The use of compact data types, such as 1-bit representations, is a current trend to enhance the effectiveness of deep neural networks. A Binary Neural Network (BNN) is a Convolutional Neural Network (CNN) of low accuracy with binarized activations and weights. BNNs normally include several layers, such as the convolutional layer, fully connected layer, pooling layer, and batch normalization layer. As shown in Figure 2a,b, an XNOR network has distinct functional blocks than CNNs. Generally, the convolutional layer, the batch normalization layer, the activation layer, and the pooling layer are just examples of the various functional layers that make up a typical neural network. The input tensor of the batch normalization layer can be normalized by computing its mean and variance. An element-wise nonlinear function (e.g., Sigmoid, ReLU) is applied to the activation layer. The pooling layer uses several pooling techniques (e.g., max, min, and average). Compared to CNN, the functional layers of the BNN are arranged differently in Figure 2b. When receiving a binarized input batch, the pooling layer suffers a significant information loss. For instance, the input batch of the min-pooling layer accepts the binarization and returns it with the majority of its members equal to -1. Thus, the grouping directly connects the convolutional layer (BinConv) to overcome the critical issue of considerable information loss. We normalize the input prior to binarization in the BNN to address the information loss problem brought on by binarization. In this case, the normalization process is efficient in increasing the model precision by forcing the input to hold a zero mean, whose threshold range shrinks to zero, resulting in a reduced binarization error. To calculate the sign(I), the binary activation layer (BinActiv) is utilized.

To combine Tensor Train Decomposition method with the binary convolution layer, we designed a structure shown in Figure 2c. The Tensor Train cores could be stored in binary format with the scaling factors for each, which is the same as in the XNOR network. After the TT reconstruction of the real weight, as demonstrated above, we would binarize it and use another scaling factor α along the real weight to perform the binary convolution. Thus, the TT-format binary convolution can be rewritten in short as:

$$\alpha_{\boldsymbol{G}}\boldsymbol{\mathcal{B}}(\alpha_0\alpha_1\ldots\alpha_a\ldots\alpha_d\boldsymbol{\mathcal{B}}(\hat{\boldsymbol{G}}_0\boldsymbol{G}_1\ldots\boldsymbol{G}_a\ldots\boldsymbol{G}_d)) = \alpha_{\boldsymbol{G}}\boldsymbol{\mathcal{B}}(\mathcal{T}) * \boldsymbol{I} \approx \sum \alpha_{\boldsymbol{G}_f}(\boldsymbol{\mathcal{B}}(\boldsymbol{T}_f) \oplus \boldsymbol{I}) \tag{8}$$

where $\boldsymbol{\mathcal{B}}$ denotes the element wise weight binarization, $\alpha_0\alpha_1\ldots\alpha_a\ldots\alpha_d$ are the scaling factor according to the Tensor Train cores , $\alpha_{\boldsymbol{G}}$ is scaling factor based on the reconstructed real weight and f stands for the filters' index in the weight $\mathcal{T}$ where $\boldsymbol{T}_f \in \mathcal{T}$, $\alpha_{\boldsymbol{G}_f} \in \alpha_{\boldsymbol{G}}$. The Tensor Train cores are fixed after the training, thus we consider the α values are fixed as well, which means it would not require the L1-Norm mean operation in FPGA implementation. Focal Loss can be used as the supervision to alleviate the effect of class imbalance. However, it only supports the standard label of the $[0,1]$ category. In order to enable the Focal Loss to train successfully on the joint model, autonomous quality focal loss is proposed, which optimizes the traditional focal loss in two parts: $-log(p_c) \rightarrow -((1-y)log(1-\sigma)+ylog(\sigma))$ and $(1-p_c)^\gamma \rightarrow |y-\sigma|^\beta$ $\beta \geq 0$. The estimation σ denotes the output of the sigmoid operators [6,42]. The modulating factor $|y-\sigma|^\beta$ means the non-negative absolute distance between y and σ. When the estimation is accurate, σ will be close to y, the loss is down-weighted. Thus, we draw less attention to the simple example. Conversely, the distance $|y-\sigma|^\beta$ hard example produced will increase, attracting more attention to learn. Our final loss is defined as: $Loss = -|y-\sigma|^\beta((1-y)log(1-\sigma)+ylog(\sigma)), \beta \geq 0$. In addition, manually setting β to a fixed value has its drawback because both a high value of β and a low value β lead to an undesirable result [2]. Inspired by automated Focal Loss [2], we design a dynamically adjusted β to fit our model convergence during the training process. The value of β should be large enough to allow the network to focus on hard samples at the beginning, then the value of β should be lower to prevent decreasing gradients. Here, we apply $1-\hat{p_c}$ and bias b to numerically alter β. Therefore, the dynamic β is defined:

$$\begin{aligned} &\beta = -log(1-\hat{p_c}) - \boldsymbol{b} \\ &where\ p_c = \begin{cases} \sigma & \text{if } y = 1 \\ 1-\sigma & \text{if } y = others \end{cases} \end{aligned} \tag{9}$$

p represents the expected probability when a sample is predicted correctly. Furthermore, the $\hat{p}$ is roughly equivalent to the mean over p in one training batch, and we use $\hat{p} = 0.95 * \hat{p}_{old} + 0.05 * \hat{p}_{new}$ here to smooth the training process. After that, we reduce the frequency of change in β to train our model effectively, and the threshold for change is set to 0.05:

$$\beta_{i+1} = \begin{cases} \beta_i & \text{if } \beta_i - \beta_{i+1} < 0.05 \\ \beta_{i+1} & \text{if } \beta_i - \beta_{i+1} \geq 0.05 \end{cases} \tag{10}$$

where i represents the iteration.

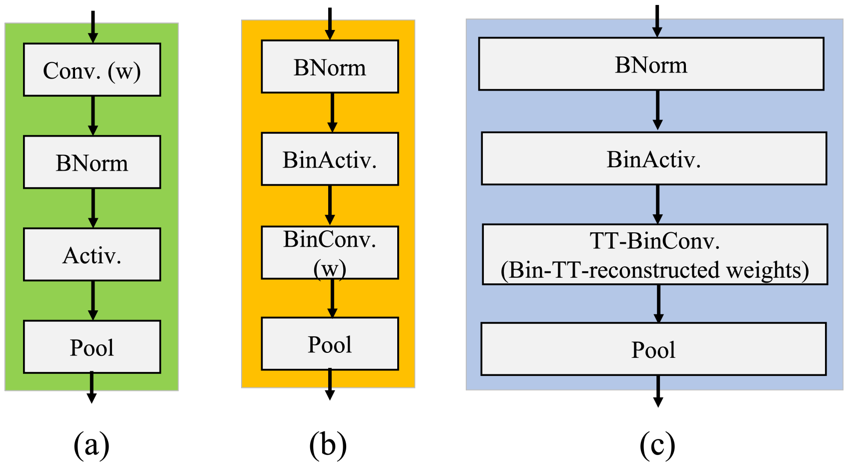

Figure 2. (**a**) Conventional CNN blocks, (**b**) XNOR-network blocks, (**c**) ABFLMC blocks.

4. Overall Hardware Architecture of ABFLMC-YOLOV5

Figure 3 shows our proposed implementation of the FPGA architecture for ABFLMC-YOLOV5. The architecture consists of a computation kernel, registers, and BRAM. Our model presents a higher compression rate resulting in a relatively friendly environment for hardware implementation. Our core weight model size has only 1.39MB in VOC dataset, which is one-fifth of the size compared to YOLOv5s. This allows fast-speed access from BRAM to become applicable. The PE (processing element) cluster is designed to address a large amount of parallel multiplication and accumulation within the convolution computation. A series of DSPs is implemented as a multiplied cumulative (MAC) system that performs the computation. Attributed to the ABFLMC-YOLOV5 algorithm, the parameters of our model were reduced to a certain level, which benefited from the scarcity of resources that is intrinsic for most low-power computing units. In Figure 3, the routing component computed by convolution is the core of the whole inference computation, which is responsible for running the overall algorithm. Schedule the timing and order of each computation and sends a control signal to access the required data. With our compressed model, the computation complexity for each layer presents an advantage when applied to a pipelining platform with sufficient hardware resources.

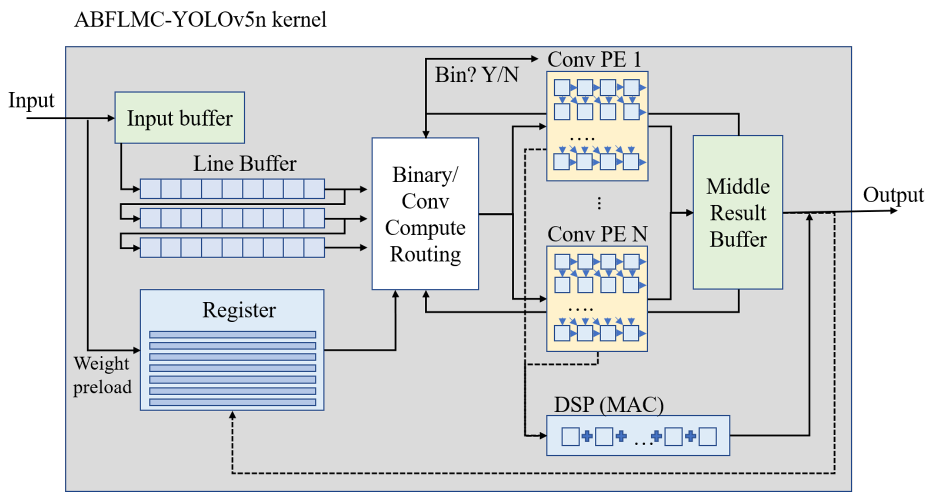

Figure 3. Overall hardware architecture of the implementation.

5. Experiment and Results

5.1. Experiment Setup

Our Yolov5n-based ABFLMC model performs the ABFLMC compression algorithm and the tensor train compression algorithm to shrink the model and simultaneously maintain accuracy. To demonstrate the effectiveness of autonomous quality focal loss in our model, we designed an ablation study based on the VOC dataset containing the Yolov5n model benchmark, the benchmark model with quality focal loss where $\gamma = 1.5$ and Tensor Train compression, and our ABFLMC model, herein we set the dynamic bias β: $b = 1.0, 1.4, 1.5$ separately, while keeping the other hyperparameters constant. Specifically, we set the epoch at 300, use rank = 16 to decompose our tensor to balance the size and precision of the model, and apply SGD optimizer with an initial learning rate of 0.01. Finally, our overall experiment is based on the PyTorch framework, and we use an Intel i9-9920X + RTX3090 PC as our hardware platform.

5.2. Ablation Study and Comparison with State of Art Models

As shown in Table 1, we use $mAP_{50:95}$, mAP_{50} and the number of parameters to evaluate the performance of the model. Obviously, compared to the original model, the size of other models after compressing the tensor train decreases from 1.79 M to 1.39 M in the VOC Dataset. In terms of accuracy, our ABFLMC model with autonomous quality focal loss performs better than the traditional Focal Loss-Tensor Train (FL-TT) model. When the bias is $b = 1.4$, our ABFLMC method achieves the best $mAP_{50:95}$ in 33.9 and mAP_{50} in 62.1. Therefore, the proposed autonomous quality focal loss architecture efficiently enhances the basic FL-TT model. Besides, Table 1 also reveals that the higher b could shrink the difference between the hard samples and easy samples, resulting in the rapid mAP drop under the higher IOU threshold. However, the higher b is helpful for the lower IOU threshold to achieve higher performance according to its 0.1 drops of mAP_{50} between $b = 1.4$ and 1.5. In contrast, lower b is more efficient in keeping mAP below the higher IOU threshold because it can enlarge the discrimination by sampling. The model keeps the similar performance of $mAP_{0.5:0.95}$ between $b = 1.0$ and 1.4. Furthermore, Table 2 shows that our ABFLMC model with $b = 1.4$ achieves a competitive result compared to other state-of-the-art light-weight models. Our model achieves the smallest size with an acceptable accuracy drop in the comparison VOC dataset based. For the COCO dataset, although the mAP_{50} of our ABFLMC model is 6 percent lower than the latest lightweight model yolov5n, our size is 22.1% less and computational complexity is 0.1 GFLOPs lower than the yolov5n. Specifically, our ABFLMC model reaches mAP_{50} of 28.1 under VisDrone Dataset, which is 2.2× higher than yolov5n while keeping the light-weight design. This comparison also reveals that our ABFLMC design achieves the advantage under the large scene and small target dataset. Therefore, our model definitely has a better trade-off between accuracy and model size.

Table 1. Ablation Study under Different Bias in VOC.

Dataset	Model	$mAP_{50:95}$	mAP_{50}	# Paras
VOC 07 + 12	Original (Yolov5n)	45.4	72.5	1.79 M
	FL-TT ($\gamma = 1.5$)	33.1	61.0	1.39 M
	ABFLMC ($b = 1.0$)	33.9	61.7	
	ABFLMC ($b = 1.4$)	33.9	62.1	
	ABFLMC ($b = 1.5$)	33.6	62.0	

Table 2. Performance Comparison Across Models and Datasets [9,30,43].

Dataset	Models	Input Size	Backbone	$mAP_{50:95}$	mAP_{50}	# Parameters (M)	GFLOPs
VOC 07 + 12	Faster R-CNN [7]	600	VGG	-	73.2	134.7	-
	Faster R-CNN [7]	600	ResNet-101	-	76.4	-	-
	R-FCN [17]	600	ResNet-101	-	79.5	50.9	-
	SSD300 [3]	300	VGG	-	75.8	26.3	-
	DSSD321 [44]	321	ResNet-101	-	78.6	>52.8	-
	GRP-DSOD320 [43]	320	DS/64-192-48-1	-	78.7	14.2	-
	YOLOv5s [9]	640	-	51.9	78.4	7.11	16.5
	TT-YOLOv5s(rank 16) [9]	640	-	48.8	76.9	4.74	18.4
	MobileNetv2-Yolov5s	640	-	50.27	76.8	4.6	10.0
	ABFLMC-YOLOv5n	640	-	33.9	62.1	1.39	4.2
COCO	CenterNet-DLA [45]	512	DLA34	39.2	57.1	16.9	52.58
	CornerNet-Squeeze [46]	511	-	34.4	-	31.77	150.15
	SSD [3]	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 [24]	320	CSPDarknet53	38.2	57.3	64.36	35.5
	YOLO-Lite [28]	224	-	12.26	-	0.6	1.0
	YOLOV3-tiny [47]	320	Tiny Darknet	14	29	8.85	3.3
	YOLOV4-tiny [24]	320	Tiny Darknet	-	40.2	6.06	4.11
	YOLObile [30]	320	CSPDarknet53	31.6	49	4.59	3.95
	YOLOv5s [48]	640	-	37.2	56.0	7.2	16.5
	TT-YOLOv5s (rank 16) [9]	640	-	34.2	54.6	4.9	18.9
	YOLOv5n [48]	640	-	28.4	46.0	1.9	4.5
	ABFLMC-YOLOv5n	640	-	22.8	40.0	1.48	4.4
VisDrone	YOLOv5n [48]	640	-	12.9	25.9	1.78	4.2
	ABFLMC-YOLOv5n	640	-	14.3	28.1	1.38	4.1

5.3. Hardware Evaluation

The hardware implementation environment is evaluated by the development evaluation board Ultrascale+ KCU116 on the XCKU5P FPGA from Xilinx. By reducing the size of the tensors and parameters with binarization, the model can be stored in the on-board storage (BRAM, regs) without using a DDR4 module. The results of the evaluation are listed in Table 3 for comparison and analysis. For the validation process of the ABFLMC model, a series of convolution computations have been applied. Thus, the process element cluster (PE) has been modified to adapt the convolution computation to speed up. With the binary computation, it only takes a small portion of computing resources compared to Float32. As a result, a mux function has been deployed in the routing of the computation in order to optimize the usage of PE speeding up the XOR computation with fewer registers and DSPs. The foremost factor affecting the on-chip power consumption will be the utilization of the Look-up Table (LUT) and Flip-Flops (FF) resources.

Table 3. Hardware evaluation in different datasets.

Hardware Evaluations	**VOC 07 + 12**	**COCO**	**VOC 07 + 12**	**COCO**
Model Name	**ABFLMC-Yolov5n**		**TT-Yolov5s** [9]	
Param M	1.39	1.48	4.74	4.9
LUT	111,233	120,533	182,022	187,022
LUT Utilization (%)	51.3	55.6	83.9	86.2
FF	174,210	178,720	123,098	143,728
FF Utilization (%)	40.2	41.2	28.4	33.1
BRAM (MB)	305	305	220	235
BRAM Utilization (%)	63.5	63.5	45.8	49
DSP	569	577	1321	1351
DSP Utilization (%)	31.2	31.6	72.4	74.1
GOPS	135.5	129.4	42.6	34.2
Power (W)	6.12	6.33	15.2	16.1

The differential function can define LUT as a register to store the active data or as a logic gate that fulfills the arithmetic requirement. In our design, LUT resources have been deployed about 51.3% for the VOC dataset and 55.56% for the COCO. The flip-flops usually work as a recording state; in most cases, they will be deployed as shared registers or high-speed buffering for the calculation. Due to the high compression rate of our model, the model size is available for pipelining design. FF utilization is approximately 40.2% for the VOC dataset and 41.2% for the COCO dataset.

The BRAM is used for storing compressed binary weight tensor cores and some float32 offset. BRAM utilization reaches 63.5% on VOC 07 + 12 and on the COCO Dataset. A dedicated MAC (Multiply Accumulate) unit has been implemented to speed up the convolution computing, which consists of multiple DSPs. In convolution computing, the addition operation can easily cause a delay due to a position replacement issue. However, DSP can be formed for a high-speed accumulator for a specific purpose to overcome the bottleneck of the computation process. Our DSP deployment shows 31.2% and 31.6% on each dataset. In Table 3, the power consumption is measured in two parts: on-chip power and off-chip power. We calculated the on-chip power for the FPGA ICs and included all peripheral devices on the board. The overall power consumption is presented in Table 3. The VOC Dataset reaches the power consumption of 6.12 W, and the GOPS reaches 135.5. The overall power consumption of the COCO Dataset is 6.33 W, and the GOPS reaches 129.4.

The state-of-the-art Yolov5 hardware comparison is presented in Table 4. However, most of the usage and result is on a different platform or did not mention in the article. We can only compare them in specific situations. With ABFLMC-YOLOv5n works, contributing to the auto focal loss and BNN quantization reducing a huge amount of the register resource utilization, the FPGA evaluation deployment can reach 6.33 W of power usage, which is only 39.3% of the TT-Yolov5s model under the COCO dataset and significantly lower than other high performance platforms. It is also difficult to monitor the power consumption of the whole operation on a mobile phone due to all the other high power-consuming parts besides the CPU (e.g., Screen, Camera). The mobile phone platform has the strength of mobility, but sacrifices the processing speed. Usually, the mobile SoC has dedicated compute units serving a specific function, only applying the general purpose compute units is quite inefficient. SVM simulation could be a possible way to enhance SoC performance according to Helali1's work [49].

Table 4. Hardware Comparison in different platforms within COCO dataset.

Models	Dataset	Platform	GFLOPs	Power (W)
ABFLMC-YOLOv5n (Ours)	COCO	Intel i9-9920X + RTX3090	4.4	200
ABFLMC-YOLOv5n (Ours)	COCO	Ultrascale + KCU116 FPGA	4.4	6.33
TT-YOLOv5s (rank16) [9]	COCO	Ultrascale + KCU116 FPGA	18.9	16.1
YOLObile [30]	COCO	Qualcomm Snapdragon 865	3.95	5 *
YOLOv5n [48]	COCO	na	4.5	na
REQ-YOLO [50]	VOC 07 + 12	ADM-7V3 FPGA	na	21

* The power is not mentioned in the article. We estimated it only based on the theoretical TDP of the SoC.

6. Conclusions

This article states a novel ABFLMC object detection model that combines efficient autonomous quality focal loss and compression of the tensor train with the acceleration of FPGA hardware. Our ABFLMC model achieves 33.9 in $mAP_{50:95}$ and 62.1 in mAP_{50} with model size 1.79M and computational complexity 4.2 GFLOPs in the VOC Dataset, 22.8 in $mAP_{50:95}$ and 40.0 in mAP_{50} with model size 1.48M and computational complexity 4.4 GFLOPs in the COCO Dataset, and 14.3 in $mAP_{50:95}$ and 28.1 in mAP_{50} with model size 1.38M and computational complexity 4.1 GFLOPs in the VisDrone Dataset. Meanwhile, our ABFLMC method also benefits the computational efficiency of hardware implementation. In the VOC 7 + 12 dataset, the throughput reaches 135.5 GOPS while the power usage remains 6.12 W. On the COCO dataset, we present 129.4 GOPS with 6.33 W. The compression ratio of the model and the reduced number of operations give high flexibility for edge computing and other low-power applications. Moreover, the TT decomposition and BNN method still have the drawback of a drop in accuracy. Our architecture still has the potential to be improved in the future such as utilizing a lighter structure or more efficient BNN method to reduce computational complexity while maintaining more competitive accuracy.

Author Contributions: Conceptualization, M.L. and Y.B.; methodology, M.L., S.L. and Y.B.; software, M.L. and K.H.; validation, M.L. and K.H.; formal analysis, M.L. and K.H.; investigation, M.L. and S.L.; resources Y.B.; data curation, M.L. and Y.B.; writing—original draft preparation, M.L., S.L., K.H. and Y.B.; writing—review and editing, M.L., S.L., K.H. and Y.B.; visualization, M.L., S.L. and K.H.; supervision, Y.B. and R.F.D.; project administration, Y.B. and R.F.D.; funding acquisition, Y.B. and R.F.D. All authors have read and agreed to the published version of the manuscript.

Funding: 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.

Data Availability Statement: The data presented in this study are available on request from the corresponding author upon reasonable request.

Acknowledgments: 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.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wu, J.; Wang, Z. A hybrid model for water quality prediction based on an artificial neural network, wavelet transform, and long short-term memory. *Water* **2022**, *14*, 610. [CrossRef]
2. Weber, M.; Fürst, M.; Zöllner, J.M. Automated focal loss for image based object detection. In Proceedings of the 2020 IEEE Intelligent Vehicles Symposium (IV), Las Vegas, NV, USA, 19 October–13 November 2020; pp. 1423–1429.
3. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 8–16 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 21–37.
4. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 580–587.
5. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft coco: Common objects in context. In Proceedings of the European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 740–755.
6. Lin, T.Y.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. Focal loss for dense object detection. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2980–2988.
7. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 91–99. [CrossRef] [PubMed]
8. Shrivastava, A.; Gupta, A.; Girshick, R. Training region-based object detectors with online hard example mining. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 761–769.
9. Liu, M.; Luo, S.; Han, K.; Yuan, B.; DeMara, R.F.; Bai, Y. An Efficient Real-Time Object Detection Framework on Resource-Constricted Hardware Devices via Software and Hardware Co-design. In Proceedings of the 2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP), Virtual Conference, 7–9 July 2021; IEEE Computer Society: Los Alamitos, CA, USA, 2021; pp. 77–84. [CrossRef]
10. Liu, M.; Han, K.; Luo, S.; Pan, M.; Hossain, M.; Yuan, B.; DeMara, R.F.; Bai, Y. An Efficient Video Prediction Recurrent Network using Focal Loss and Decomposed Tensor Train for Imbalance Dataset. In Proceedings of the 2021 on Great Lakes Symposium on VLSI, Virtual Event, 22–25 June 2021; pp. 391–396.
11. Comon, P. Tensor decompositions, state of the art and applications. *arXiv* **2009**, arXiv:0905.0454.
12. De Lathauwer, L.; De Moor, B.; Vandewalle, J. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.* **2000**, *21*, 1253–1278. [CrossRef]
13. Viola, P.; Jones, M. Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2001, Kauai, HI, USA, 8–14 December 2001; Volume 1; p. I.
14. Dalal, N.; Triggs, B. Histograms of oriented gradients for human detection. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 20–26 June 2005; IEEE: Piscataway, NJ, USA, 2005; Volume 1, pp. 886–893.
15. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
16. Girshick, R. Fast r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1440–1448.
17. Dai, J.; Li, Y.; He, K.; Sun, J. R-fcn: Object detection via region-based fully convolutional networks. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 379–387.
18. Qiao, S.; Chen, L.C.; Yuille, A. Detectors: Detecting objects with recursive feature pyramid and switchable atrous convolution. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 19–25 June 2021; pp. 10213–10224.
19. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
20. Redmon, J.; Farhadi, A. YOLO9000: Better, faster, stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 7263–7271.
21. Redmon, J. Darknet: Open Source Neural Networks in C. 2013. Available online: https://pjreddie.com/darknet/ (accessed on 31 August 2022).
22. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1026–1034.
23. Miller, G.A.; Beckwith, R.; Fellbaum, C.; Gross, D.; Miller, K.J. Introduction to WordNet: An on-line lexical database. *Int. J. Lexicogr.* **1990**, *3*, 235–244. [CrossRef]
24. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. Yolov4: Optimal speed and accuracy of object detection. *arXiv* **2020**, arXiv:2004.10934.
25. Zaidi, S.S.A.; Ansari, M.S.; Aslam, A.; Kanwal, N.; Asghar, M.; Lee, B. A Survey of Modern Deep Learning based Object Detection Models. *arXiv* **2021**, arXiv:2104.11892.

26. Jocher, G.; Stoken, A.; Borovec, J.; Christopher, S.T.A.N.; Laughing, L.C. ultralytics/yolov5: V4.0—nn.SiLU() Activations, Weights & Biases Logging, PyTorch Hub Integration. 2021. Available online: https://zenodo.org/record/4418161 (accessed on 31 August 2022).
27. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520.
28. Huang, R.; Pedoeem, J.; Chen, C. YOLO-LITE: A real-time object detection algorithm optimized for non-GPU computers. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018; pp. 2503–2510.
29. Li, Y.; Li, J.; Lin, W.; Li, J. Tiny-DSOD: Lightweight object detection for resource-restricted usages. *arXiv* **2018**, arXiv:1807.11013.
30. Cai, Y.; Li, H.; Yuan, G.; Niu, W.; Li, Y.; Tang, X.; Ren, B.; Wang, Y. Yolobile: Real-time object detection on mobile devices via compression-compilation co-design. *arXiv* **2020**, arXiv:2009.05697.
31. Hoff, P.D. Equivariant and scale-free Tucker decomposition models. *Bayesian Anal.* **2016**, *11*, 627–648. [CrossRef]
32. Rai, P.; Wang, Y.; Guo, S.; Chen, G.; Dunson, D.; Carin, L. Scalable Bayesian low-rank decomposition of incomplete multiway tensors. In Proceedings of the International Conference on Machine Learning, PMLR, Bejing, China, 22–24 June 2014; pp. 1800–1808.
33. Zhao, Q.; Zhang, L.; Cichocki, A. Bayesian CP factorization of incomplete tensors with automatic rank determination. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *37*, 1751–1763. [CrossRef]
34. Ermis, B.; Cemgil, A.T. A Bayesian tensor factorization model via variational inference for link prediction. *arXiv* **2014**, arXiv:1409.8276.
35. Jørgensen, P.J.; Nielsen, S.F.; Hinrich, J.L.; Schmidt, M.N.; Madsen, K.H.; Mørup, M. Probabilistic parafac2. *arXiv* **2018**, arXiv:1806.08195.
36. Zheng, Y.; Xu, A.B. Tensor completion via tensor QR decomposition and L2, 1-norm minimization. *Signal Process.* **2021**, *189*, 108240. [CrossRef]
37. Oseledets, I.V. Tensor-train decomposition. *SIAM J. Sci. Comput.* **2011**, *33*, 2295–2317. [CrossRef]
38. Deng, C.; Sun, F.; Qian, X.; Lin, J.; Wang, Z.; Yuan, B. TIE: Energy-efficient tensor train-based inference engine for deep neural network. In Proceedings of the 46th International Symposium on Computer Architecture, Phoenix, AZ, USA, 22–26 June 2019; pp. 264–278.
39. Novikov, A.; Podoprikhin, D.; Osokin, A.; Vetrov, D. Tensorizing neural networks. *arXiv* **2015**, arXiv:1509.06569.
40. Novikov, A.; Izmailov, P.; Khrulkov, V.; Figurnov, M.; Oseledets, I.V. Tensor Train Decomposition on TensorFlow (T3F). *J. Mach. Learn. Res.* **2020**, *21*, 1–7.
41. Garipov, T.; Podoprikhin, D.; Novikov, A.; Vetrov, D. Ultimate tensorization: Compressing convolutional and fc layers alike. *arXiv* **2016**, arXiv:1611.03214.
42. Tian, Z.; Shen, C.; Chen, H.; He, T. Fcos: Fully convolutional one-stage object detection. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Korea, 27–28 October 2019; pp. 9627–9636.
43. Shen, Z.; Shi, H.; Feris, R.; Cao, L.; Yan, S.; Liu, D.; Wang, X.; Xue, X.; Huang, T.S. Learning object detectors from scratch with gated recurrent feature pyramids. *arXiv* **2017**, arXiv:1712.00886.
44. Fu, C.Y.; Liu, W.; Ranga, A.; Tyagi, A.; Berg, A.C. Dssd: Deconvolutional single shot detector. *arXiv* **2017**, arXiv:1701.06659.
45. Duan, K.; Bai, S.; Xie, L.; Qi, H.; Huang, Q.; Tian, Q. Centernet: Keypoint triplets for object detection. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Korea, 27–28 October 2019; pp. 6569–6578.
46. Law, H.; Teng, Y.; Russakovsky, O.; Deng, J. Cornernet-lite: Efficient keypoint based object detection. *arXiv* **2019**, arXiv:1904.08900.
47. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
48. Jocher, G.; Stoken, A.; Chaurasia, A.; Borovec, J.; Kwon, Y.; Michael, K.; Liu, C.; Fang, J.; Abhiram, V.; Chaurasia, A.; et al. ultralytics/yolov5: V6.0—YOLOv5n 'Nano' Models, Roboflow Integration, TensorFlow Export, OpenCV DNN Support. Zenodo. 2021. Available online: https://zenodo.org/record/5563715 (accessed on 31 August 2022).
49. Helali, A.; Ameur, H.; Górriz, J.; Ramírez, J.; Maaref, H. Hardware implementation of real-time pedestrian detection system. *Neural Comput. Appl.* **2020**, *32*, 12859–12871. [CrossRef]
50. Ding, C.; Wang, S.; Liu, N.; Xu, K.; Wang, Y.; Liang, Y. REQ-YOLO. In Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, 24–26 February 2019. [CrossRef]