electronics

MDPI

Article

Hybrid CNN-SVM Inference Accelerator on FPGA Using HLS

Bing Liu, Yanzhen Zhou, Lei Feng *, Hongshuo Fu and Ping Fu

School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin 150000, China; liubing66@hit.edu.cn (B.L.); 20s005077@stu.hit.edu.cn (Y.Z.); 21s105135@stu.hit.edu.cn (H.F.); fuping@hit.edu.cn (P.F.)

* Correspondence: hitfenglei@hit.edu.cn

Abstract: Convolution neural networks (CNN), support vector machine (SVM) and hybrid CNN-SVM algorithms are widely applied in many fields, including image processing and fault diagnosis. Although many dedicated FPGA accelerators have been proposed for specific networks, such as CNN or SVM, few of them have focused on CNN-SVM. Furthermore, the existing accelerators do not support CNN-SVM, which limits their application scenarios. In this work, we propose a hybrid CNN-SVM accelerator on FPGA. This accelerator utilizes a novel hardware-reuse architecture and unique computation mapping strategy to implement different calculation modes in CNN-SVM so that it can realize resource-efficient acceleration of the hybrid algorithm. In addition, we propose a universal deployment methodology to automatically select accelerator design parameters according to the target platform and algorithm. The experimental results on ZYNQ-7020 show that our implementation can efficiently map CNN-SVM onto FPGA, and the performance is competitive with other state-of-the-art works.

Keywords: convolution neural network (CNN); support vector machine (SVM); field-programmable gate array (FPGA); hybrid algorithm accelerator; computation mapping; design space exploration (DSE); high-level synthesis (HLS)

Citation: Liu, B.; Zhou, Y.; Feng, L.; Fu, H.; Fu, P. Hybrid CNN-SVM Inference Accelerator on FPGA Using HLS. *Electronics* **2022**, *11*, 2208. https://doi.org/10.3390/electronics11142208

Academic Editors: Bruno Da Silva and Jo Vliegen

Received: 21 May 2022
Accepted: 11 July 2022
Published: 14 July 2022

1. Introduction

Machine learning algorithms, such as CNN and SVM, have achieved great success in image classification, target detection and other fields. Hybrid CNN-SVM algorithms have also been extensively adopted in medical image recognition [1] and fault diagnosis [2], etc. With the recent development of edge computing and the Internet of Things (IoT), a growing number of application scenarios in embedded systems require algorithms to be processed in real time and with low power consumption. FPGAs have been widely used in these scenarios to deploy algorithms due to the high energy efficiency and reconfigurability.

Recently, many dedicated FPGA-based accelerators have been proposed for CNN or SVM algorithms [3–5]. They can achieve high performance in CNN or SVM acceleration. However, they are not suitable for other CNN-based or SVM-based hybrid algorithms, such as CNN-SVM or CNN-RNN. This severely limits the application scope of these accelerators. Proposing a hybrid CNN-SVM accelerator is challenging and valuable because CNN and SVM each has unique calculation modes and resource requirements.

Currently, the research of hybrid algorithm accelerators has also gradually gained attention. The authors in [6] designed high-performance RTL IPs of CNNs and RNNs to accelerate their hybrid algorithm; however, the incompatibility of IPs caused insufficient utilization of hardware resources, and it is difficult for the RTL design to follow the rapid development of algorithms. The work [7] transformed different calculation modes of CNN into a uniformed matrix-multiplication before acceleration. However, the convolution in CNN requires data reuse. Therefore, there will be a large amount of data duplication leading to bandwidth waste because of the transformation. Work [8,9] used the same hardware resource to implement different calculation modes in hybrid algorithms.

As in [6,7], they all lack guidance for deploying algorithms on specific platforms efficiently in practical applications. The work [3,10] made an in-depth and comprehensive design of a CNN accelerator based on FPGA, which was not suitable for SVM. The authors in [11,12] studied the SVM FPGA accelerator; however, they accelerated SVM computation in simple hardware implementation.

Our work proposes a novel FPGA-based hybrid CNN-SVM accelerator and deployment method, which provides a reference for the design of other hybrid algorithm accelerators. The underlying general operator and computation mapping strategy are applied in the accelerator to be compatible with the different calculation modes in CNN and SVM on the same hardware resource. Depending on such a hardware-reuse method, our accelerator greatly improves resource utilization and reduces bandwidth requirements. In addition, our work is flexible and scalable owing to using high-level-synthesis (HLS) technology.

The main contributions of this work are:

1. A novel computing architecture based on hardware reuse is designed for compatibility with hybrid CNN-SVM. We apply a resource-efficient general operator to implement typical calculation modes, such as convolution in CNN and decision functions in SVM.
2. A computation mapping strategy is adapted for transforming SVM to spatial convolution. In this way, the accelerator can efficiently reuse the computing logic resource to complete the inference calculations of CNN-SVM.
3. A universal deployment methodology is provided that combines the uniformed representation of CNN and SVM with the computing architecture of our accelerator. According to the target platform and algorithm, it supports searching the optimal implementation parameters for CNN, SVM or CNN-SVM through design space exploration.
4. A typical CNN-SVM hybrid network is implemented with the proposed computing architecture and computation mapping strategy. It can achieve a high performance with 13.33 GOPs and 0.066 NTP with very few resources, which outperforms other state-of-the-art methods.

2. Architecture and Realization

In terms of the calculation principles of CNN and SVM, the convolution calculation in CNN and the decision function calculation in SVM can be decomposed into matrix multiplication or matrix–vector multiplication in the underlying implementation. The hybrid CNN-SVM accelerator proposed in this section reconstructs the calculation of SVM and transforms it into the spatial convolution. Therefore, the accelerator can efficiently implement different calculation modes of CNN and SVM.

2.1. Hybrid CNN-SVM Accelerator Architecture

The proposed CNN-SVM accelerator architecture is shown in Figure 1. In order to fully exploit the hardware resources, we use the same computing logic resources to implement the core computing parts of CNN and SVM (i.e., convolution in CNN and the decision function in SVM), and the underlying calculation is supported by the resource-efficient general operator.

Moreover, considering the scalability of the accelerator, other parts of CNN and SVM (i.e., pooling, activation functions and voting), which are incompatible with the core computing parts and require very few resources, are implemented separately.

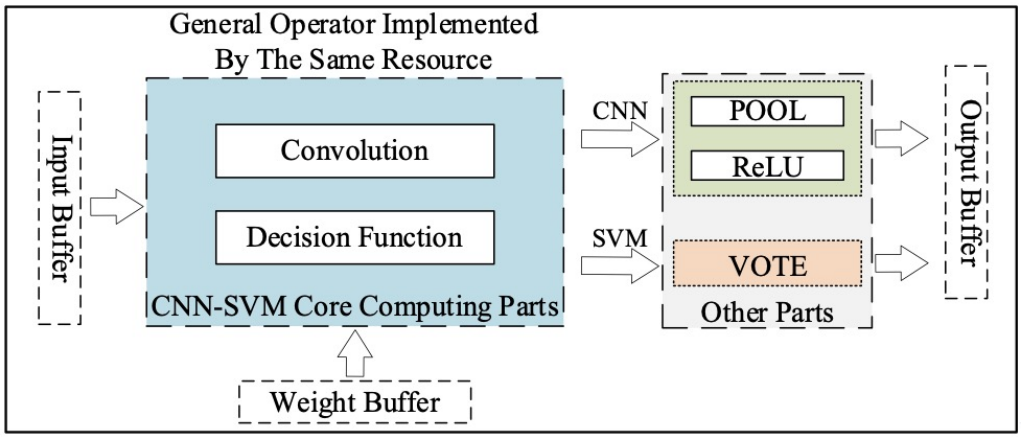

Figure 1. Overall architecture of the hybrid CNN-SVM accelerator.

2.2. Resource-Efficient General Operator

In order to be compatible with the computation-intensive convolution in CNN and the communication-intensive decision function in SVM, our hybrid CNN-SVM accelerator use the tiled-convolution method [4] to construct the basic processing element (PE) and adopts a corresponding computation mapping strategy for transforming the decision function into convolution. Consequently, it enables CNN and SVM to reuse the underlying PE to achieve high performance and resource utilization.

The general operator is shown in Figure 2. It is optimized in two dimensions with the parallelisms of input channel (Tn) and output channel (Tm). First, it takes Tn elements as input from the convolution feature map or reconstructed SVM feature vector in each clock cycle. Second, the input elements will multiply and accumulate with the corresponding weight through a multiply-accumulate (MAC) tree of size $Tn \cdot Tm$. Finally, different calculations, such as bias accumulation, pooling, activation functions and SVM voting, is performed with the constraint of the current layer type.

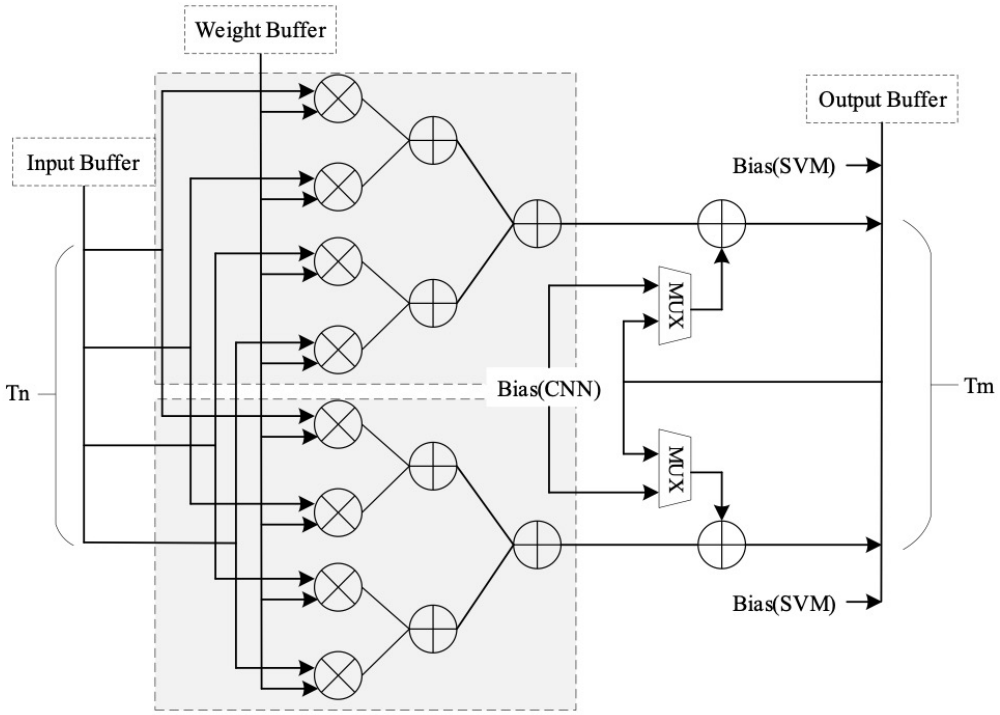

Figure 2. Resource-efficient general operator.

2.3. General Implementation of CNN-SVM

Typical calculation modes of CNN include operations, such as convolution, activation functions and pooling [9]. Convolution is the core block of CNN during feed-forward computation. It receives several input feature maps and applies 2-D convolution with kernel weights and bias to generate local output feature maps as in Equation (1):

$$OutputFM_j = \sum_{i=1}^{N} InputFM_i * Weight_{ij} + bias_j \quad (1)$$

where $InputFM_i$ and $Weight_{ij}$ represent the i-th input feature map and the convolution kernel weights applied on the i-th input map to generate the j-th output feature map, respectively.

Then, a nonlinear activation function is applied on every pixel of output feature maps to improve the expressive ability of CNN. The rectified linear unit (ReLU) as shown in Equation (2) is the most frequently adopted function in CNN.

$$ReLU(x) = max(x, 0) \quad (2)$$

Pooling is adopted in the middle of a continuous convolution layer for down-sampling and to avoid over-fitting. It outputs the maximum (max pooling) or average (average pooling) value of each subarea in the input feature map.

The implementation of CNN based on a general operator is shown in Figure 3. First, the input feature map is divided into $< Tr, Tc, Tn >$, where Tr, Tc and Tn are the tiling factors of the row (R_i), column (C_i) and input channel (N) in the feature map, respectively. The partitioned CNN feature map is multiplied and accumulated with the corresponding kernel weight $< Kr, Kc, Tn >$. Then, the activation function and pooling operation are performed to obtain the sub-block $< Tx, Ty, Tm >$ of the output feature map $< R_o, C_o, M >$. As CNN is generally stacked by CONV-ReLU-POOL, our implementation is aimed at this structure. Depending on the target CNN network, ReLU and POOL can also be flexibly replaced by sigmoid, tanh, global average pooling and so on.

In this working mechanism, PE is repeatedly reused until the specified layer is completed, thereby, realizing the standard calculation mode of CNN. In addition, our implementation has a certain versatility for large-scale CNN networks due to the tiled-convolution processing.

The core part of SVM is the decision function (Figure 4. Equation (1)). Considering the characteristics of parallel computing on FPGA, the kernel function $K(x_i, x)$ implemented in this work is a linear kernel as shown in Figure 4. Equation (2). Figure 4 shows that we can combine the weights (α_i, γ_i) into a matrix (W) during inference calculation and convert them into multiplication and accumulation operations of W matrix and I vector, which can be regarded as the calculation mode of matrix–vector multiplication as shown in Figure 4. (N_{svm} and M_{svm} in Figure 4 represent the dimensions of the input and output feature vectors of SVM.)

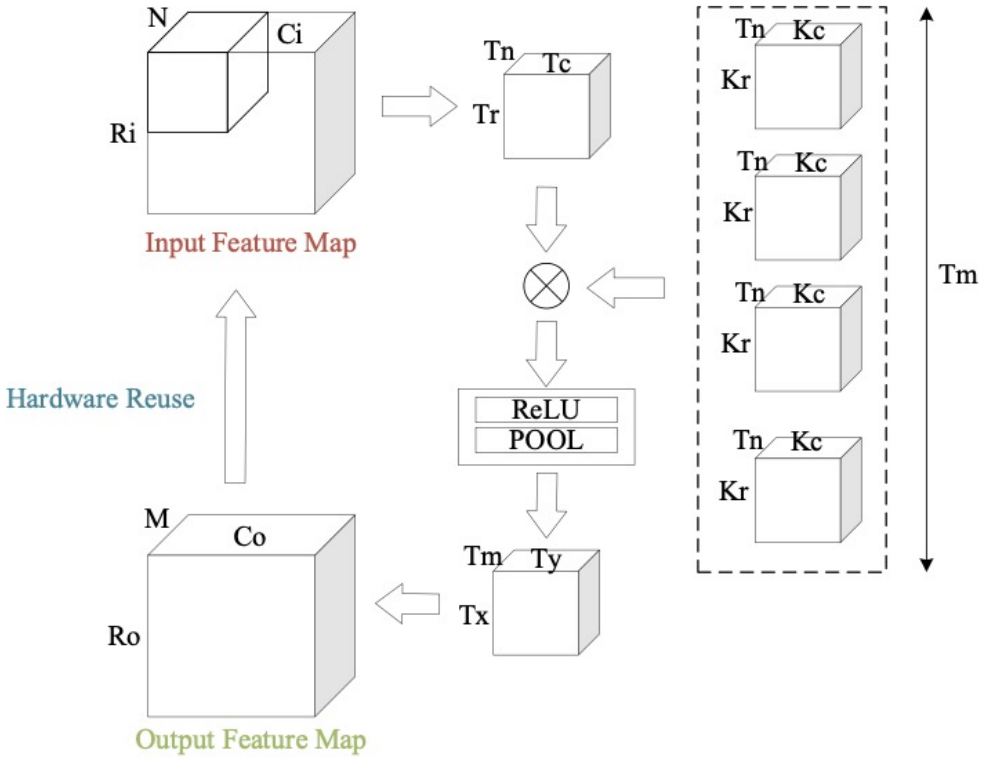

Figure 3. Implementation diagram of CNN.

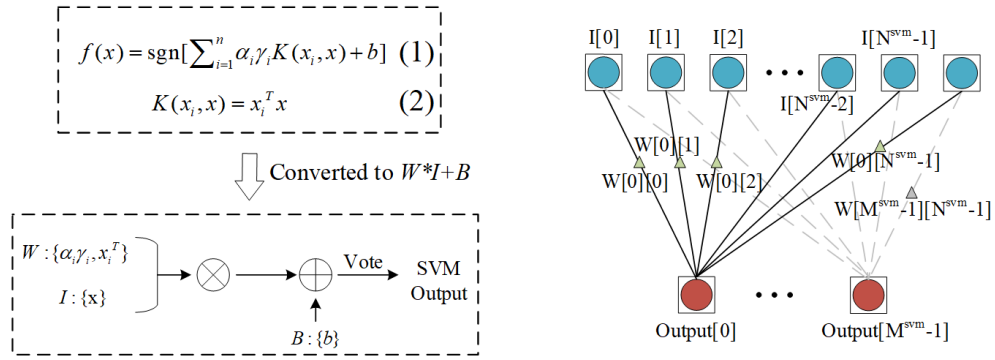

Figure 4. Convert decision function into matrix–vector multiplication.

As shown in Figure 5, when mapping the transformed decision function to spatial convolution, two optimized computation mapping strategies are proposed: I-FM (input to feature map) and K-FM (kernel to feature map). In order to maximize the utilization rate of hardware resources, both strategies are intensive-mapping, which means that the mapped SVM vectors or weights almost filling all the feature maps ($Ri \cdot Ci$) and filters ($Kr \cdot Kc$). In addition, these two strategies incorporate batch processing to improve the data reuse rate and reduce the dependence on bandwidth during SVM calculations.

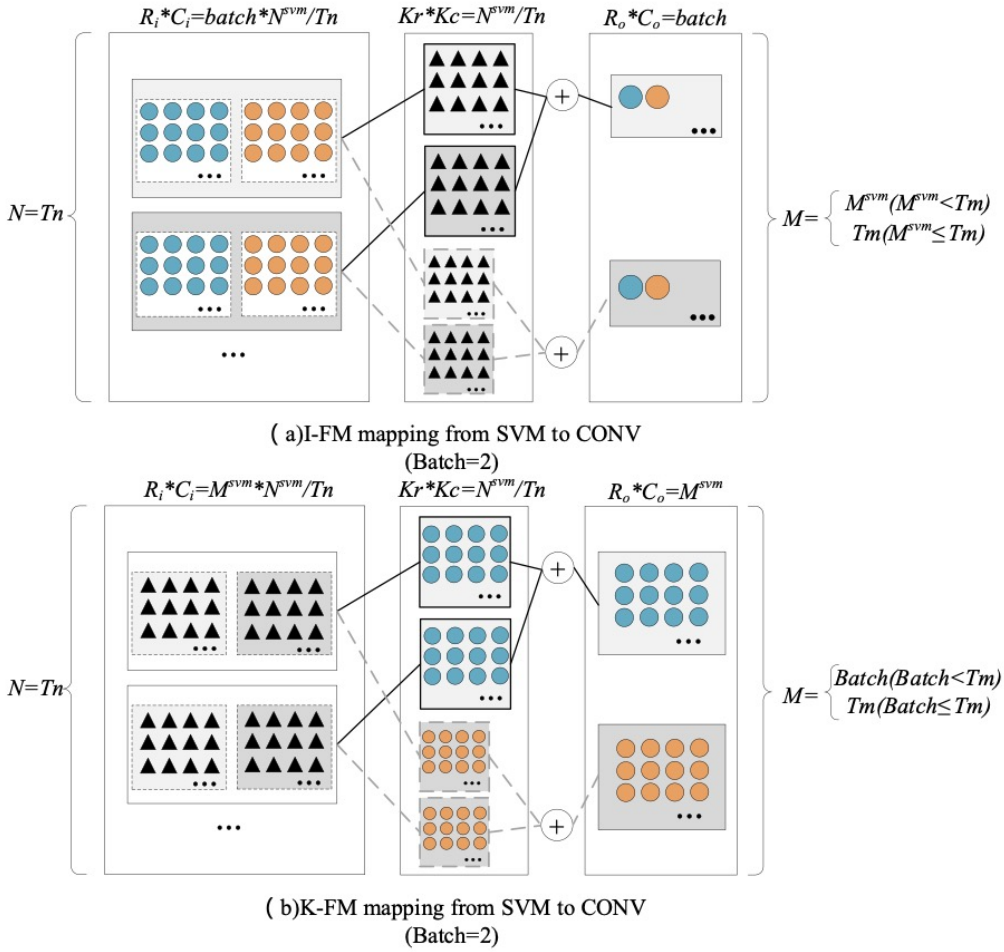

Figure 5. Computation mapping strategies of SVM.

I-FM strategy shown as Figure 5a maps the input vector (I) of SVM to the CNN input feature map, and the kernel weight matrix (W) is mapped to the convolution kernel. Clearly, the weight reuse rate of this strategy is *batch*; however, *batch* should not be overly large because of the restriction of Tr and Tc.

The K-FM strategy shown as Figure 5b maps the kernel weight matrix (W) to the CNN input feature map, and the input vector (I) of SVM is mapped to the convolution kernel. The weight reuse rate of K-FM is also related to *batch*. However, since K-FM performs multi-batch processing on the output channel, *batch* in this strategy is not limited. To maximize the utilization of DSP, the recommended *batch* selection is equal to or close to Tm, and thus the *batch* numbers of the outputs can be obtained in one period. Moreover, this strategy can also reduce the number of reusing processing elements and data overload. Therefore, if *batch* is greater than Tn, K-FM is more efficient than I-FM.

The implementation process of SVM is shown in Figure 6. First, the calculation of SVM is converted into the matrix–vector multiplication in Figure 4, and then it is mapped to spatial convolution through the computation mapping strategy (I-FM/K-FM). In addition, if multi-classification issues are involved, voting is also required. By this means, the general operator can be efficiently reused for implementing SVM.

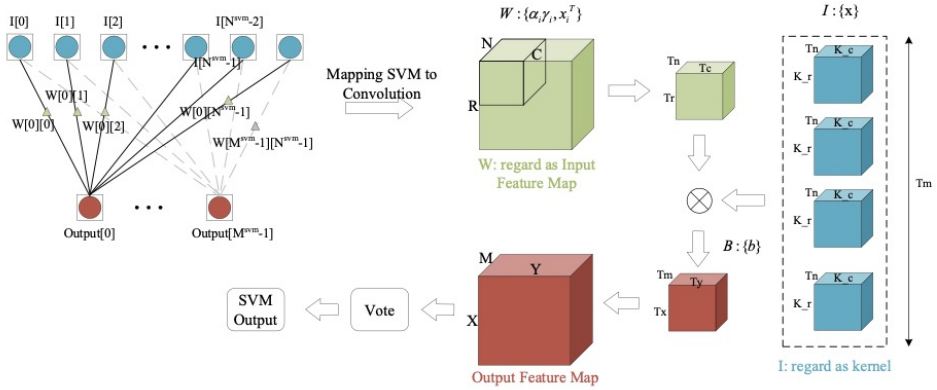

Figure 6. Implementation of SVM.

3. Universal Deployment Methodology

In order to efficiently map a hybrid CNN-SVM algorithm onto an FPGA-based accelerator, we propose a universal deployment methodology. Given the target algorithm structure and platform hardware resource constraints, it can search the optimal architecture design parameters (tiling factors and SVM-mapping parameters) through design space exploration.

3.1. Uniformed Representation

For the sake of making this methodology not only suitable for CNN-SVM but also for the individual design of CNN or SVM, we use a uniformed representation for CNN and SVM in Table 1, which also facilitates the subsequent design space exploration. Benefiting from this representation, our method can also be easily extended to other hybrid algorithms, such as CNN-RNN. Furthermore, considering the difficulty of SVM-mapping strategies selection under different *batch* size and hardware resource constraints, we will conduct a quantitative experiment in Section 4 to compare I-FM and K-FM.

Table 1. Uniformed representation parameters for CNN and SVM.

Network Parameters	Uniformed Representation	CNN	SVM	
			I-FM	K-FM
Input FM	$R_i \cdot C_i$	$R_i^{conv} \cdot C_i^{conv}$	$Batch \cdot N^{svm}/Tn$	$M^{svm} \cdot N^{svm}/Tn$
Output FM	$R_o \cdot C_o$	$R_o^{conv} \cdot C_o^{conv}$	$Batch$	M^{svm}
Input Channel	N	N^{conv}	Tn	Tn
Output Channel	M	M^{conv}	$M^{svm}(Tm)$	$Batch(Tm)$
Kernel	$K_r \cdot K_c$	$K_r \cdot K_c$	N^{svm}/Tn	N^{svm}/Tn
Stride	$S_r \cdot S_c$	$S_r \cdot S_c$	N^{svm}/Tn	N^{svm}/Tn

3.2. Resource Evaluation Model

To meet the resource constraints of target platform and performance requirements, we combined the uniformed representation in Table 1 to construct a corresponding evaluation model for the resource consumption of our hybrid CNN-SVM accelerator. This model can be used to estimate the consumption of computing resources (DSPs) and memory resources (BRAMs), which we most care about in the FPGA-based algorithm accelerator. In order to facilitate the general deployment of CNN and SVM, this resource evaluation model is similar to the evaluation model in most present studies of CNN accelerators.

The parallelism of operator in accelerator is $Tm \cdot Tn$. Since DSP blocks in FPGA are used to implement multiplication and addition operations with HLS, the DSP resource consumption is:

$$DSP = Tm \cdot Tn \cdot (MUL_{dsp} + ADD_{dsp}) \quad (3)$$

where MUL_{dsp} and ADD_{dsp} are the number of DSPs required for multiplication and addition, which are different under various levels of data accuracy (such as FP32: MUL_{dsp} = 3, ADD_{dsp} = 2).

The on-chip Block RAMs (BRAMs) consumed by the accelerator are divided into three parts: input buffer, weight buffer and output buffer. Input buffer stores the partitioned CNN feature map or mapped SVM vector, and the block size is $Tr \cdot Tc \cdot Tn$. The weight buffer stores the corresponding kernel weight ($Kr \cdot Kc \cdot Tn \cdot Tm$). The output buffer is divided into two parts to store the results of PEs (multiplication and accumulation) and post-processing (activation function, pooling and SVM-bias accumulation), and the size is $(Tr \cdot Tc \cdot Tm + Tr/2 \cdot Tc/2 \cdot Tm)$. Thus, the total BRAM resources used in the accelerator are:

$$\begin{aligned} BRAM = & \frac{CEILING(T_r \cdot T_c, 2)}{512} \cdot (T_n + T_m) \\ & + \frac{CEILING(K_r \cdot K_c, 2)}{512} \cdot (T_n \cdot T_m) \\ & + \frac{CEILING(T_r/2 \cdot T_c/2, 2)}{512} \cdot T_m \end{aligned} \tag{4}$$

where $CEILING(X, 2)$ means that X is a multiple of 2, and the minimum is 512, which depends on the characteristic of HLS.

3.3. Design Space Exploration

The central issue of design space exploration is how to select the architecture design parameters in Table 1 to maximize the accelerator performance under hardware resource constraints. We turn this into a constrained optimization problem to solve. The objective of the optimization problem is to minimize latency as Equation (5), which mainly focuses on the computation-intensive multiply and accumulate operations:

$$\frac{\frac{R}{Tr} \cdot \frac{C}{Tc} \cdot \frac{M}{Tm} \cdot \frac{N}{Tn} \cdot (Tr \cdot Tc \cdot Kr \cdot Kc + Const)}{frequency} \tag{5}$$

where R, C, M and N are the length and width of the feature map and the number of output and input channels, respectively.

Subject to:

$$DSP = Tm \cdot Tn \cdot (MUL_{dsp} + ADD_{dsp}) \leq DSP_{total} \tag{6}$$

$$BRAM = InputBuffer + WeightBuffer + OutputBuffer \leq BRAM_{total} \tag{7}$$

$$0 \leq Tr \cdot Tc \leq (M^{svm} \cdot N^{svm}/Tn) \tag{8}$$

$$0 \leq Tr \leq max\{R_i^{conv}, R_i^{svm}\} \tag{9}$$

$$0 \leq Tc \leq max\{C_i^{conv}, C_i^{svm}\} \tag{10}$$

$$0 \leq Tm \leq M \tag{11}$$

$$0 \leq Tn \leq N \tag{12}$$

where Equations (6) and (7) are the resource constraints of DSP and BRAM. Equations (8)–(12) are the constraints on the tiling factors $< Tr, Tc, Tm, Tn >$, which relate to the network structure, BRAM capacity and SVM-mapping strategy. Moreover, the purpose of constraints Equations (8)–(12) is to perform intensive SVM-mapping to make full use of BRAM resources. In the process of design space exploration, we exhaust all the design parameters (tiling factors: Tr, Tc, Tm and Tn) that may meet the constraints (Equations (6)–(12)) and then select the optimal optimization objective (i.e., the minimum latency as shown in Equation (5)).

4. Evaluation

4.1. Environment Setup

We evaluate the versatility of the proposed accelerator and deployment methodology with a common combination structure of CNN-SVM algorithm. Table 2 describes the structure of this algorithm. We implement the CNN-SVM algorithm with the proposed hybrid CNN-SVM accelerator and compare our design with three state-of-the-art FPGA-based accelerators [4,13,14]. In addition, considering the limited resources in most embedded devices and to verify the effectiveness of our proposed deployment methodology, a Xilinx ZYNQ-7020 FPGA board was selected as our target platform due to its limited numbers of DSP and BRAM (220 and 280). The CNN-SVM shown in Table 2 is implemented on 200 MHz.

Table 2. Structure of the CNN-SVM algorithm.

	Layer1	**Layer2**	**Layer3**	**Layer4**
Type	CNN	CNN	CNN	SVM
R_i/C_i	28/28	14/14	7/7	1/1
N	1	4	8	256
M	4	8	16	45

4.2. Design Space Exploration

Combining the resource constraints of ZYNQ-7020 and the CNN-SVM network structure, as well as the proposed deployment methodology in Section 3, the optimal tiling factors $< Tr, Tc, Tm, Tn >$ can be obtained as $< 36, 40, 16, 8 >$ through design space exploration.

In addition, in order to select the SVM computation mapping strategy (I-FM/K-FM), we conduct an experiment to compare I-FM with K-FM under different *batch* sizes. The SVM-mapping parameters in Table 1 can be calculated with $< Tr, Tc, Tm, Tn >$. Tables 3 and 4 show the actual resource utilization and performance evaluated in execution cycles of these two strategies. K-FM is more suitable for multi-batch processing, and it can effectively utilize BRAMs.

Table 3. Resource utilization under different SVM mapping strategies.

Strategy	**Resource**			
	BRAM	**DSP**	**FF**	**LUT**
I-FM	40	201	28,728	38,307
K-FM	64	203	28,396	38,071

Table 4. Clock cycles under different batches.

Strategy	**Batch Size**			
	1	**8**	**16**	**32**
I-FM	7703	8567	17,130	34,256
K-FM	7512	7953	8541	17,074

4.3. Results and Comparison

Table 5 compares our hybrid CNN-SVM accelerator with the CNN accelerator [4] and the hybrid network accelerators [13,14]. In Table 5, the SVM is mapped in K-FM strategy. From the comparison results, it can be seen that our accelerator has fully utilized the computing resources (92%) of ZYNQ-7020 through design space exploration. However, due to the limitation of DSPs, our throughput (GOPS) is relatively low. For a fair comparison, we use the normalized throughput (NTP = GOPS/DSP) [15] to measure the performance of the accelerator. Our NTP is higher than in other works. This is because our accelerator com-

pletes the efficient mapping between different calculation modes in the hybrid algorithm, which can effectively utilize on-chip resources to achieve high computation efficiency.

Therefore, under the same resource constraints, our accelerator can achieve a higher throughout, which proves that our accelerator is competitive with the state-of-the-art works. Compared to the dedicated accelerators, our design not only has more applicability but also the calculation speed of single CNN or SVM is comparable. Moreover, it is worth mentioning that our accelerator can deploy algorithms quickly and effectively in actual embedded application scenarios due to the proposed universal deployment methodology.

Table 5. Performance evaluation and comparison with other works.

	[4]	[13]	[14]	**Ours**
Device	VX485T	VX690T	VC709	ZYNQ7020
Frequency	100 MHz	250 MHz	100 MHz	200 MHz
Precision	32 bit	8 bit	12 bit	16 bit
Total DSP	2800	3600	3600	220
Used DSP	2240	3104	3130	203
Utilization	80%	86%	87%	92%
Power(W)	18.61	\	23.6	2.03
GOPs	61.62	13.74	36.25	13.33
NTP	0.027	0.004	0.012	0.066
Normalize	1.0	0.15	0.44	2.44

5. Conclusions

In this work, we proposed a hybrid CNN-SVM FPGA-based accelerator. We adopted a new hardware-reuse architecture and computation mapping strategy to be compatible with CNN and SVM calculations. We demonstrated that the accelerator can effectively map a CNN-SVM algorithm onto FPGA and that it will be easily applied through our deployment methodology. We used very few on-chip resources (203 DSPs) to achieve the throughput of 13.33 GOPs and higher NTP (0.066), which are much higher than those of other studies. In addition, through our universal deployment strategy, we can map different types of CNN, SVM and hybrid CNN-SVM algorithms onto different target FPGA platforms. Therefore, our accelerator has the advantages of scalability and usability. In the future, we will integrate network compression technologies, such as data quantification, into our accelerator and consider more standard computing modes or algorithms, such as depthwise separable convolution and recurrent neural networks.

Author Contributions: Conceptualization, B.L. and Y.Z.; Background, B.L., Y.Z. and P.F.; Methodology, Y.Z.; Validation, Y.Z. and H.F.; Investigation, B.L., Y.Z. and L.F.; Resources, B.L., P.F. and L.F.; Writing—original draft preparation, B.L. and Y.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China, grant number 62171156.

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

Nomenclature

CNN	Convolution neural network
SVM	Support vector machine
$I-FM, K-FM$	Mapping strategies of SVM
Ri, Ci, N	Row, column and input channel of input feature map
Ro, Co, M	Row, column and channels of output feature map
Kr, Kc	Size of convolution kernel
Sr, Sc	Size of kernel moving stride in row and column

Tr, Tc	Tiling width and height of input feature map
Tm, Tn	Parallelisms of input channel and output channel
Tx, Ty	Size of sub-block in output feature map

References

1. Xue, D.X.; Zhang, R.; Feng, H.; Wang, Y.L. CNN-SVM for microvascular morphological type recognition with data augmentation. *J. Med. Biol. Eng.* **2016**, *36*, 755–764. [CrossRef] [PubMed]
2. Kang, J.; Park, Y.J.; Lee, J.; Wang, S.H.; Eom, D.S. Novel leakage detection by ensemble CNN-SVM and graph-based localization in water distribution systems. *IEEE Trans. Ind. Electron.* **2018**, *65*, 4279–4289. [CrossRef]
3. Guo, K.; Sui, L.; Qiu, J.; Yu, J.; Wang, J.; Yao, S.; Han, S.; Wang, Y.; Yang, H. Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2017**, *37*, 35–47. [CrossRef]
4. Zhang, C.; Li, P.; Sun, G.; Guan, Y.; Xiao, B.; Cong, J. Optimizing fpga-based accelerator design for deep convolutional neural networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2015.
5. Dass, J.; Narawane, Y.; Mahapatra, R.N.; Sarin, V. Distributed Training of Support Vector Machine on a Multiple-FPGA System. *IEEE Trans. Comput.* **2020**, *69*, 1015–1026. [CrossRef]
6. Zeng, S.; Guo, K.; Fang, S.; Kang, J.; Xie, D.; Shan, Y.; Wang, Y.; Yang, H. An Efficient Reconfigurable Framework for General Purpose CNN-RNN Models on FPGAs. In Proceedings of the 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP), Shanghai, China, 19–21 November 2018; pp. 1–5.
7. Suda, N.; Chra, V.; Dasika, G.; Mohanty, A.; Ma, Y.; Vrudhula, S.; Seo, J.S.; Cao, Y. Throughput-optimized openCL-based FPGA accelerator for large-scale convolutional neural networks. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 21–23 February 2016; pp. 16–25.
8. Sun, Y.; Liu, B.; Xu, X. An OpenCL-Based Hybrid CNN-RNN Inference Accelerator On FPGA. In Proceedings of the 2019 International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, 9–13 December 2019; pp. 283–286.
9. Li, H.; Fan, X.; Jiao, L.; Cao, W.; Zhou, X.; Wang, L. A high performance FPGA-based accelerator for large-scale convolutional neural networks. In Proceedings of the Field Programmable Logic and Applications (FPL), 2016 26th International Conference on IEEE, Lausanne, Switzerland, 29 August–2 September 2016.
10. Wu, D.; Zhang, Y.; Jia, X.; Tian, L.; Li, T.; Sui, L.; Xie, D.; Shan, Y. A high-performance CNN processor based on FPGA for MobileNets. In Proceedings of the 2019 29th International Conference on Field Programmable Logic and Applications (FPL), Barcelona, Spain, 9–13 September 2019.
11. Afifi, S.; GholamHosseini, H.; Sinha, R. FPGA implementations of SVM classifiers: A review. *SN Comput. Sci.* **2020**, *1*, 1–17. [CrossRef]
12. Ons, B.; Bahoura, M. Efficient FPGA-based architecture of an automatic wheeze detector using a combination of MFCC and SVM algorithms. *J. Syst. Archit.* **2018**, *88*, 54–64.
13. Yin, S.; Tang, S.; Lin, X.; Ouyang, P.; Tu, F.; Liu, L.; Wei, S. A high throughput acceleration for hybrid neural networks with efficient resource management on FPGA. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **2019**, *38*, 678–691. [CrossRef]
14. Zhang, X.; Liu, X.; Ramachran, A.; Zhuge, C.; Tang, S.; Ouyang, P.; Cheng, Z.; Rupnow, K.; Chen, D. High-performance video content recognition with long-term recurrent convolutional network for FPGA. In Proceedings of the 2017 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, Belgium, 4–8 September 2017.
15. Ma, Y.; Cao, Y.; Vrudhula, S.; Seo, J.S. Automatic compilation of diverse CNNs onto high-performance FPGA accelerators. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **2020**, *39*, 424–437. [CrossRef]