electronics

MDPI

Article

Accelerating Event Detection with DGCNN and FPGAs

Zhe Han, Jingfei Jiang *, Linbo Qiao, Yong Dou, Jinwei Xu and Zhigang Kan

National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha 410073, China; hanzhe18@nudt.edu.cn (Z.H.); qiao.linbo@nudt.edu.cn (L.Q.); yongdou@nudt.edu.cn (Y.D.); xujinwei13@nudt.edu.cn (J.X.); kanzhigang5206@163.com (Z.K.)

* Correspondence: jingfeijiang@nudt.edu.cn

Received: 30 August 2020; Accepted: 5 October 2020; Published: 13 October 2020

Abstract: Recently, Deep Neural Networks (DNNs) have been widely used in natural language processing. However, DNNs are often computation-intensive and memory-expensive. Therefore, deploying DNNs in the real world is very difficult. In order to solve this problem, we proposed a network model based on the dilate gated convolutional neural network, which is very hardware-friendly. We further expanded the word representations and depth of the network to improve the performance of the model. We replaced the Sigmoid function to make it more friendly for hardware computation without loss, and we quantized the network weights and activations to compress the network size. We then proposed the first FPGA (Field Programmable Gate Array)-based event detection accelerator based on the proposed model. The accelerator significantly reduced the latency with the fully pipelined architecture. We implemented the accelerator on the Xilinx XCKU115 FPGA. The experimental results show that our model obtains the highest F1-score of 84.6% in the ACE 2005 corpus. Meanwhile, the accelerator achieved 95.2 giga operations (GOP)/s and 13.4 GOPS/W in performance and energy efficiency, which is 17/158 times higher than the Graphics Processing Unit (GPU).

Keywords: FPGA; event detection; dilate gated convolutional neural network; accelerator

1. Introduction

With the rapid development of the Internet, it has become more and more important to extract usefully structured information from massive amounts of unstructured texts. The Information Extraction (IE) tasks aim to identify event descriptions (including entities, relationships, and events) from the unstructured natural text, to classify them into predefined categories, and to store them in a structured form for users to query and further analyze. Event Detection (ED), which aims to accurately identify event triggers of specific types, is an important and challenging part of IE tasks. For example, a "movement" event triggered by "force" should be extracted from the following texts "A wildfire in California forced hundreds of people from their homes".

Event detection systems today are commonly based on pattern matching [1–4] and machine learning (ML) [5–8]. Pattern matching achieves high performance in a particular domain but less portability. Whenever the system is ported to a new scenario, new patterns must be built. Tuning patterns is a time-consuming process and requires considerable experience. Meanwhile, machine learning does not require much guidance from domain experts and has better portability. With the increasing abundance of various textual resources on the Internet, the corpus is no longer the bottleneck for machine learning. At present, machine learning has become the main research method for event detection.

The ML-based approach relies on ML algorithms and deep learning models to identify and classify trigger words. Specifically, methods such as a support vector machine, Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs) are used to determine whether a word is the trigger. They focus on how to learn representation of the sentence in the text to improve the accuracy of identification and classification. The advantage of this approach is that it requires little manpower and knowledge of new domains. Recently, neural networks have been widely used in event detection [9,10], which encodes the semantics of context in a high-dimensional feature space. CNN [11–13] and RNN [14,15] are the most commonly used neural network models for event detection today.

CNNs have gained great success and show dramatic performance in many tasks like image recognition [16,17], video analysis [18,19], and natural language processing (NLP) [11,12]. Collobert et al. [20] applied CNN to NLP tasks and succeeded in avoiding task-specific engineering as early as 2011. Instead, they relied on large unlabeled data sets and let the training algorithm discover internal representations. Xia and Liu [21] further tested the performance on a Chinese event detection task using a similar model and demonstrated the potential of CNNs for event detection tasks. However, classical CNN may miss valuable facts when considering multiple-event sentences. Chen et al. [22] proposed the dynamic multi-pooling convolutional neural network (DMCNN), which uses a dynamic multi-pooling layer to reserve more crucial information. Lin et al. [23] further developed DMCNN on Chinese and tried to solve the word-trigger mismatch problem by directly proposing entire trigger nuggets centred at each character regardless of word boundaries. RNNs offer the capability of recognizing long-term dependencies in sequential and temporal data. They are very suitable for NLP naturally and have demonstrated state-of-the-art performance in various NLP tasks, notably machine translation [24], Part-of-Speech (PoS) tagging [25], and named-entity recognition [26]. Based on Long Short-Term Memory (LSTM) networks, Tai et al. [27] proposed Tree-LSTM, a generalization of LSTMs to tree-structured network topologies. The results suggested that Tree-LSTM is more effective for ED tasks compared to the classic bidirectional LSTM model. However, LSTMs also face the problem of word trigger mismatch for Chinese. Thus, Ding et al. [28] proposed the trigger-aware lattice neural network, which dynamically incorporated word and character information to avoid mismatch problem. Xi et al. [29] proposed a similar method to improve character-wise models by incorporating word information and language model representation into Chinese character representation.

However, the computational cost and memory requirements significantly increased with the accuracy improved, which restricted the applications of CNNs and RNNs. Therefore, high-performance Central Processing Units (CPUs) and Graphics Processing Units (GPUs) have been widely used to process CNN and RNN models, which can achieve 10 tera operations (TOPs) per second on a chip. However, large scale clusters with hundreds or thousands of computation nodes are still needed to achieve inference performance of millions of words per second. For example, DMCNN [22] takes 1.0 giga operations (GOPs) to process a single sentence with 30 words. Besides, it is quite difficult for general-purpose processors to accomplish RNN computation in parallel due to irregular computation and memory access. CPU and GPU platforms still have disadvantages of high power and low performance per watt for complex CNN and RNN models. It means that a high-performance accelerator is highly desired. Thus, various accelerators have been proposed recently. Considering the performance, energy efficiencym and flexibility, it is obvious that FPGA (Field Programmable Gate Array)-based accelerators are a sensible approach for complex CNN and RNN models.

Currently, there are many high-performance accelerators for images recognizing tasks [30,31] and video processing tasks [32,33] but no accelerators for NLP tasks. On the one hand, NLP tasks also have to process huge amounts of data just like images processing and video processing tasks. On the other hand, some NLP tasks such as question-answering systems also require low latency. Furthermore, all the current models only focus on the accuracy of the tasks, without considering the obstacles brought by the enormous computational cost and memory requirements. To address

these issues, we present the design of a Chinese event detection accelerator for FPGAs. In order to reduce the computation and footprint, we design a novel CNN model based on multilayer Dilate Gated Convolutional Neural Network (DGCNN). EE-DGCNN [34] demonstrated the potential of the DGCNN for event detection tasks and FPGA implementations. DGCNN can reduce computation complexity while obtaining long-term dependencies owing to the use of the dilated convolution. This paper makes the following major contributions:

1. To our best knowledge, we are the first to study FPGA acceleration for NLP tasks. We present a CNN model which overwhelms the previous event detection works on ACE 2005 Chinese corpus.
2. We improve the computational efficiency of the model on hardware by optimizing the activation function and quantizing the model.
3. We implement our event detection accelerator on FPGA and show significant improvements over CPU and GPU baselines.

The paper is organized as follows. Section 2 introduces the basic background of event detection and the design directions of the accelerator. Section 3 presents our Chinese event detection model. Section 4 shows our hardware-oriented model optimizing strategy. Section 5 introduces the architecture of the accelerator. Section 6 reports the experimental results, and Section 7 concludes the paper.

2. Background

2.1. Event Detection

The event detection task in this paper was defined in Automatic Content Extraction (ACE) [35] evaluations. The event consists of an event trigger and an event argument. The event detection task mainly aims to find event triggers and to categorize events to predefined event types. To help understand the task, we first introduce some event extraction terms.

- Event mention: The description of a specific event, including trigger words and argument.
- Event type: The specific predefined category of an event.
- Event trigger: Keywords or phrases that clearly express the occurrence of an event.
- Event arguments: The participants or attributes of the event.

The ACE 2005 evaluation has 8 types of events and 33 subtypes. In this paper, we ignore the hierarchical structure and make 33 subtypes as 33 types of events. Besides, we add a NONE type to predefined event types for non-trigger words.

2.2. Convolutional Neural Networks

Convolutional neural networks generally consist of various network layers such as the convolutional layer, the fully connected layer, and the pooling layer. In particular, convolutional layers are the heart of the convolutional neural network, which often occupy the main part of the computation of the entire convolutional neural network. Therefore, it is critical for accelerating the convolutional neural network to simplify the convolutional computation or to design an appropriate hardware architecture to fit the computation. Methods such as the frequency domain convolution and Winograd algorithm are widely used. The convolution can be categorized into one-dimensional (1-D) convolution, two-dimensional (2-D) convolution, and others based on the dimension. The 1-D convolution is often used in sequence problems, such as NLP tasks. The 2-D convolution is usually used in image processing tasks, such as object recognition. However, there is a difference in the ED task. For traditional ED models, the researches tend to use a whole sentence as an input to extract the dependencies of adjacent words. Thus, they have to perform convolution operations within and between words at the same time and they use a 2-D convolution instead of a 1-D convolution.

For simplicity, we introduce the 2-D convolution in ED tasks with a sentence that contains l words. Assuming that each word is represented by a vector of length w, the sentence constitutes a 2-D matrix

of the shape $[l, w]$. Suppose the kernel with the shape of $[k, w]$ is used for the convolution operation, where k is the height of the kernel. The convolutional window generates a series of sets of features by sliding between words. By multiplying and summing each set of features with the convolution kernel, we get the final result as shown in Figure 1. We use X for the input sentence, Y for the output, and W for the kernel. Assuming the stride size of 1, the results are computed as the Equation (1).

$$Y_i = \sum_{j=i}^{i+k-1} (X_j \bigotimes W) + bias \tag{1}$$

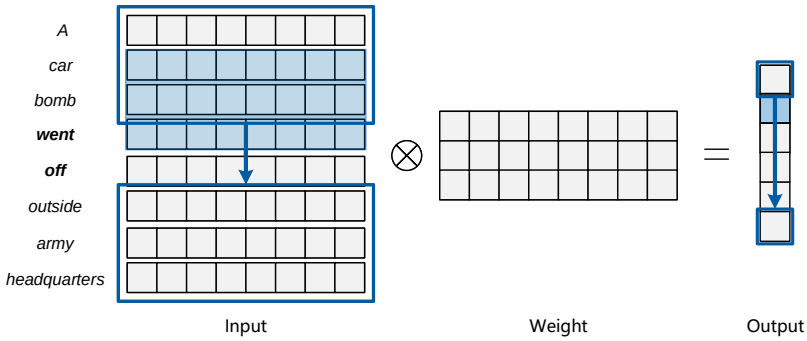

Figure 1. Two-dimensional (2-D) convolution operations: applying 2-D convolution on a sentence. When processing the Chinese corpus, each Chinese character has an embedding representation, just like an English word.

2.3. Dilated Convolutional Neural Networks

Dilated convolutional neural networks were first applied to the neural network by Yu and Koltun [36] for solving the imperfect match between classical convolutional neural networks and semantic segmentation requirements. Compared to classical convolution, the dilated convolution adds an extra option for the kernel: the dilation. If the dilation is 1, the convolution is the same as a normal convolution. If the dilation is 2, then a zero will be inserted between every two elements of the kernel. In general, suppose the dilation is n, then $n - 1$ zeroes will be inserted between every two elements of the kernel.

Figure 2 shows the information extraction capability of normal convolution and dilated convolution with the same size of kernel and three convolutional layers. The nodes in the third layer of normal convolution can obtain information from four nodes in the first layer at the maximum, while the nodes in the third layer of dilated convolution can obtain information from eight nodes in the first layer at the maximum. Moreover, the parameters and computations are exactly the same as those of normal convolution. There are two main advantages of dilated convolution.

- The receptive field will expand as the dilatation rate increases.
- The number of computations and parameters do not change with the dilatation rate.

Normal convolutional neural networks generally perform worse than recurrent neural networks (e.g., LSTM) in some NLP tasks [37]. A major reason is that the receptive field of the convolutional neural network is limited by the size of the kernel. In contrast, recurrent neural networks naturally have access to information over long distances. In general, each output feature can only capture information from input features covered by kernels. There is no possibility to capture information from long distances in the normal convolutional layer. For information at longer distances, the convolutional kernel must be expanded. For 1-D convolution, the number of parameters and computations is linearly related to the kernel size, while for 2-D convolution and other high-dimensional convolution, expanding the kernel means the amount of computations explodes. The appearance of dilated convolutional neural networks greatly alleviates this problem. By adjusting the dilatation rate,

convolutional neural networks gained the perception of long-term dependence similar to recurrent neural networks while keeping the characteristic of higher parallelizability than recurrent neural networks. Therefore, dilated convolution is more suitable for NLP tasks as well as hardware acceleration than normal convolution.

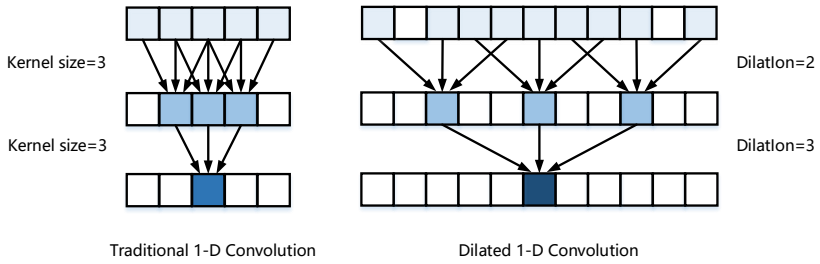

Figure 2. Traditional convolutional neural networks vs. dilated convolutional neural networks.

2.4. Word Representation

Words are usually represented using vectors in NLP tasks. Early researchers used to encode words with one-hot, but this approach cannot reflect the inner connection between words and it often leads to the curse of dimensionality when facing a large corpus of text. Word2vec [38] was proposed to solve these problems by mapping words to a vector space. It greatly reduced the data dimension, and word vectors that share common contexts were located close to one another in the space. However, they only had one representation for one word. It was helpless to deal with the polysemy. Therefore, Peters et al. [39] proposed Embeddings from Language Models (ELMo), which uses bidirectional LSTM to obtain the contextual information of each word and generates representation for every word. OpenAI applied a similar idea and proposed the OpenAI GPT (generative pretrained transformer) [40]. They used a transformer instead of the traditional bidirectional LSTM which ELMo used. However, OpenAI GPT used unidirectional transformers and only the previous contextual information was actually obtained. Therefore, Google [41] proposed BERT (Bidirectional Encoder Representations from Transformers). BERT performed amazingly well on SQuAD (The Stanford Question Answering Dataset) 1.1 and achieved the best performance in 11 different NLP tasks. Later, Facebook [42] further optimized it and developed RoBERTa (Robustly optimized BERT approach), which uses a larger model with more training data and outperforms BERT.

2.5. EE-DGCNN

EE-DGCNN was one of the best models in event extraction recently. It achieved the best performance in ACE 2005 corpus with the help of the DGCNN. Its computation complexity was much smaller than other models such as standard CNN and bidirectional LSTM. EE-DGCNN consisted of a 12-layer DGCNN and a fully connected layer. The DGCNN is a new structure based on the dilated convolution combined with the gated linear unit and residual connection. Gehring et al. [43] first proposed this structure and applied it to the NLP task. Figure 3 shows the basic structure of the DGCNN. It can be calculated by Equation (2):

$$Y = 1DConv_1(X) \bigotimes \sigma(1DConv_2(X)) \tag{2}$$

where Y is the output result, X is the input vector sequence, σ is the Sigmoid function, and $1DConv_1$ and $1DConv_2$ are two 1-D convolutions with the same output size. This structure can avoid the vanishing of gradients during training. On the one hand, it can accelerate the flow of information and improve the performance of the model. We have implemented the EE-DGCNN model with the help of Kan and Qiao and designed a Chinese event detection model based on it.

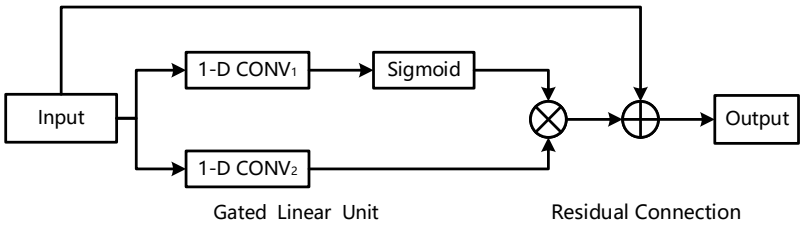

Figure 3. Basic unit of dilated gated convolutional neural network.

3. Model Design

From Section 2, the dilated CNNs can obtain long-term dependencies in NLP tasks while the standard CNNs cannot. Although dilated CNNs cause irregularities in accessing memory during computation, we can avoid it by designing FPGA hardware accelerators with a custom hardware architecture. Inspired by EE-DGCNN, we chose DGCNN as the centre of our event detection network. The performance of EE-DGCNN has demonstrated that DGCNN has adequate potential for event extraction. However, there are two major limitations of the EE-DGCNN in Chinese event detection.

EE-DGCNN was originally designed to extract events for English. It lacks sufficient classification ability when migrating to Chinese. We trained the EE-DGCNN model directly on ACE 2005 corpus. Its performance is comparable to the previous work. The detailed results are shown in Table 1.

Table 1. Comparison of Chinese event detection.

Model	Precision	Recall	F1-Score
DMCNN [22]	61.6	58.8	60.2
C-BiLSTM [7]	60	60.9	60.4
NPN [23]	60.9	69.3	64.8
TLNN [28]	64.5	71.5	67.8
Bi-LSTM+CRF [29]	66.4	76	70.9
BERT-CRF [44]	no report	no report	83.2
EE-DGCNN	90.5	60.8	72.7

Like most of the neural network models, the EE-DGCNN was not designed for hardware. For example, the gated mechanism contained in the EE-DGCNN requires the Sigmoid function as a gated function, but the Sigmoid function consumes huge resources to calculate in the hardware.

To address these, we modify the network structure based on EE-DGCNN to make it more suitable for the Chinese event detection task and the hardware implementation. The biggest differences compared to the EE-DGCNN are the following.

Wider word vector representation. Compared to English, event extraction studies on Chinese are more difficult. The English event extraction task can capture more additional information at the lexical level than Chinese. For example, English words are separated while Chinese lacks natural delimiters. We can tell singular and plural by word form and part-of-speech tagging. English verbs have morphological changes, and verbal nouns are easily distinguished from general verbs.

Therefore, we decided to expand the word representation with more linguistic features for improving event detection performance. In the original EE-DGCNN, the authors used BERT to encode the input text. When we used EE-DGCNN to process the Chinese language, we chose the Chinese BERT model with a 12-layer transformer and 768 hidden units. However, the experimental result was not satisfying. We decided to use the Chinese RoBERTa model with 1024 hidden units to encode the Chinese text. We retrained EE-DGCNN based on RoBERTa and as shown in Table 2. The results showed that using RoBERTa can significantly improve the classification performance of the model.

Deeper network. Unlike other CNN models (such as DMCNN), the EE-DGCNN model was a character-wise classification model. Therefore, when we expanded the dimension of the word representation, the receptive field was not enough to cover the whole word. There are two options

to solve this problem: increasing the dilation or network depth. Obviously, expanding the dilation can ensure that the receptive field is large enough to reach the entire word vector for one hidden unit. However, large dilations may cause some information missing and may bring obstacles for future hardware parallelism, especially for small capacity embedded hardware. Therefore, we chose to increase the depth of the network for the performance of our model. Table 3 shows the performance of the model with various depths.

Table 2. Comparison of Bidirectional Encoder Representations from Transformers (BERT) and Robustly optimized BERT approach (RoBERTa).

Model	Embeddings	Precision	Recall	F1-Score
EE-DGCNN	BERT	90.5	60.8	72.7
EE-DGCNN	RoBERTa	93	68.4	78.8

Table 3. Comparison of different network depths.

Model	Embeddings	Depth	Precision	Recall	F1-Score
EE-DGCNN	RoBERTa	13	93	68.4	78.8
Model1	RoBERTa	16	93.6	68.5	79.1
Model2	RoBERTa	19	94.1	70.8	80.8
Model3	RoBERTa	22	95.5	69.5	80.5
Model4	RoBERTa	25	95.6	73.1	82.8
Model5	RoBERTa	28	94.2	72.7	82.1

4. Model Optimization

4.1. Optimization of Dilation

The dilation is a key parameter of the dilated convolution. It directly affects the accuracy of the network, but at the same time, it also determines the locality of reference of the network, which is crucial for the performance of hardware computations. The accelerator speeds up the computation by increasing the parallelism, which requires the memory to provide more than one input simultaneously. However, the Block Random Access Memory (RAM) on the FPGA chip can only provide two R/W ports at most. We have to divide the memory when the parallelism is greater than 2. EE-DGCNN used three dilations: 1, 2, and 5. Note that 5 is not a power of 2, which means that the memory (buffers) needs to be divided into 5n parts. However, the size of both the Block RAM and our data is divisible by 2. Therefore, using such dilations will inevitably waste memory resources to satisfy the data throughput rate. Otherwise, the pipeline must be stalled to reduce the requirements for data throughput rates, which further slows down the accelerator. Consequently, we modified the original dilations and retrained the model to keep the performance of the model. The experiments show that adjusting the dilations to 1, 2, and 4 can achieve better performance than 1, 2, and 5, and the results are shown in Table 4.

Table 4. Comparison of different dilation.

Model	Embeddings	Depth	Dilation	Precision	Recall	F1-Score
Model4	RoBERTa	25	1,2,5	95.6	73.1	82.8
Model6	RoBERTa	25	1,2,4	94.8	74.5	83.4

4.2. Simplification of Sigmoid

In DGCNN, the Sigmoid function is necessary for processing the output of the dilated convolution. The formula for the Sigmoid function is expressed as Equation (3).

$$Sigmoid(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{3}$$

The Sigmoid function contains exponential operations and division operations, which are expensive for hardware implementation. We optimized the Sigmoid function from the viewpoint of accelerator implementation and network model to observe the effect of Sigmoid function on the performance of the model. Firstly, we approximated the Sigmoid function using the Range Addressable Lookup Table method. We can get the Sigmoid result directly by looking up the Block Random Access Memory (BRAM). Experiments showed that the output of the Sigmoid function only requires 8-bit for hardware calculation. The specific strategy is as follows.

(1) When the input is greater than 4, the output result is 1.

(2) When the input is greater than or equal to -4 and less than or equal to 4, use the 2 integer digits of the input and 6 decimal digits as the address to get the result of the Sigmoid.

(3) When the input is less than -4, the output result is 0.

The approximated Sigmoid function is shown in Figure 4.

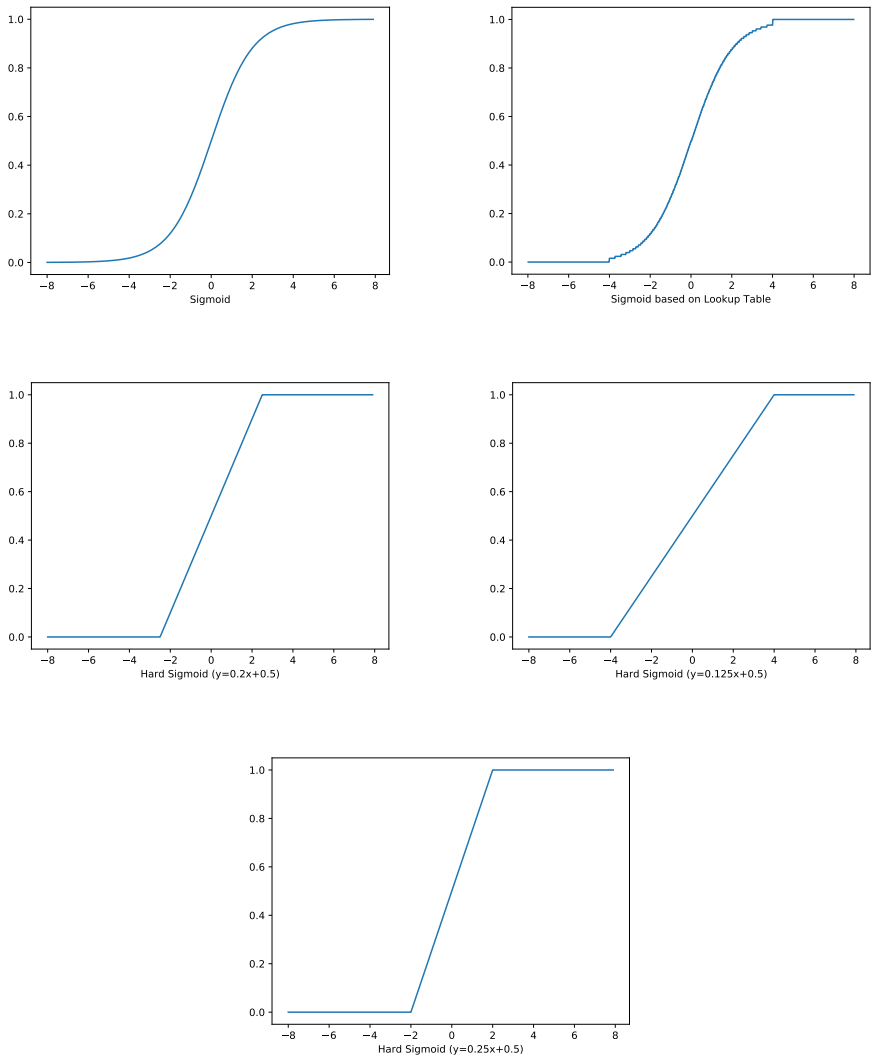

Figure 4. Sigmoid functions and hardware-friendly approximations.

On the other hand, we carefully studied the structure of the DGCNN network and concluded that the Sigmoid function is just used to provide a threshold value and is not irreplaceable. Therefore, we replaced the Sigmoid function with the more hardware-friendly Hard Sigmoid function. It is calculated by Equation (4):

$$f(x) = 0.2x + 0.5 \tag{4}$$

Considering that sard Sigmoid still requires a multiplication operation and an addition operation, we further modified the Hard Sigmoid function by modifying the factor 0.2 of x to 0.125 and 0.25, as shown in Figure 4. The advantage of using these two factors is that the multiplication can be done only by a shift operation, which greatly reduces the area usage of the chip. We retrained the modified DGCNN and the results satisfied our conjecture. The F1-score with the Hard Sigmoid function of the factor 0.125 improved by about 1% over standard Sigmoid, as shown in Table 5.

Table 5. Comparison of different gated functions.

Model	Embeddings	Depth	Dilation	Gated Function	Factor	Precision	Recall	F1-Score
Model6	RoBERTa	25	1,2,4	Sigmoid	-	94.8	74.5	83.4
Model7	RoBERTa	25	1,2,4	Sigmoid(LUT)	-	94.2	74.4	83.1
Model8	RoBERTa	25	1,2,4	Hard Sigmoid	0.2	95.2	74.9	83.8
Model9	RoBERTa	25	1,2,4	Hard Sigmoid	0.125	96.1	75.6	84.6
Model10	RoBERTa	25	1,2,4	Hard Sigmoid	0.25	95.6	74.4	83.7

4.3. Quantization

DGCNNs dramatically reduced the computational complexity, but all data representations were using 32-bit floating-point, consuming much more resources. Recent works [45] demonstrated that most neural networks do not require 32-bit in the inference process, and a 16-bit data bit width is usually sufficient to hold the accuracy. In this paper, we quantized floating-point numbers into fixed-point numbers and reduced the data bit width to the minimum while maintaining the accuracy. This can achieve higher parallelism under the same area and power consumption while further reducing the pressure on bandwidth and memory resources.

The quantization strategy was determined by the data distribution and was tested by software simulation. Figure 5 shows the value distribution histogram. In this paper, various combinations of data precision are tested, and it is finally determined that the weights and inputs of the network need only 8 bits to meet the precision requirements, while the data and intermediate results need 16 bits to keep the network precision. Table 6 shows the F1-score losses with quantization.

We finally design an event detection model suitable for hardware implementation through the above design methods. The final network structure is shown in Table 7.

Table 6. The loss of quantization on F1-score.

Target	Bitwidth	Bitwidth for Integer	Loss
Input	8	6	−0.03
Feature	16	8	−0.0006
Conv Kernel	8	2	<−0.01%
Bias	8	3	<−0.01%
FC Weight	8	2	<−0.01%
FC (Fully Connected) Bias	8	1	<−0.01%

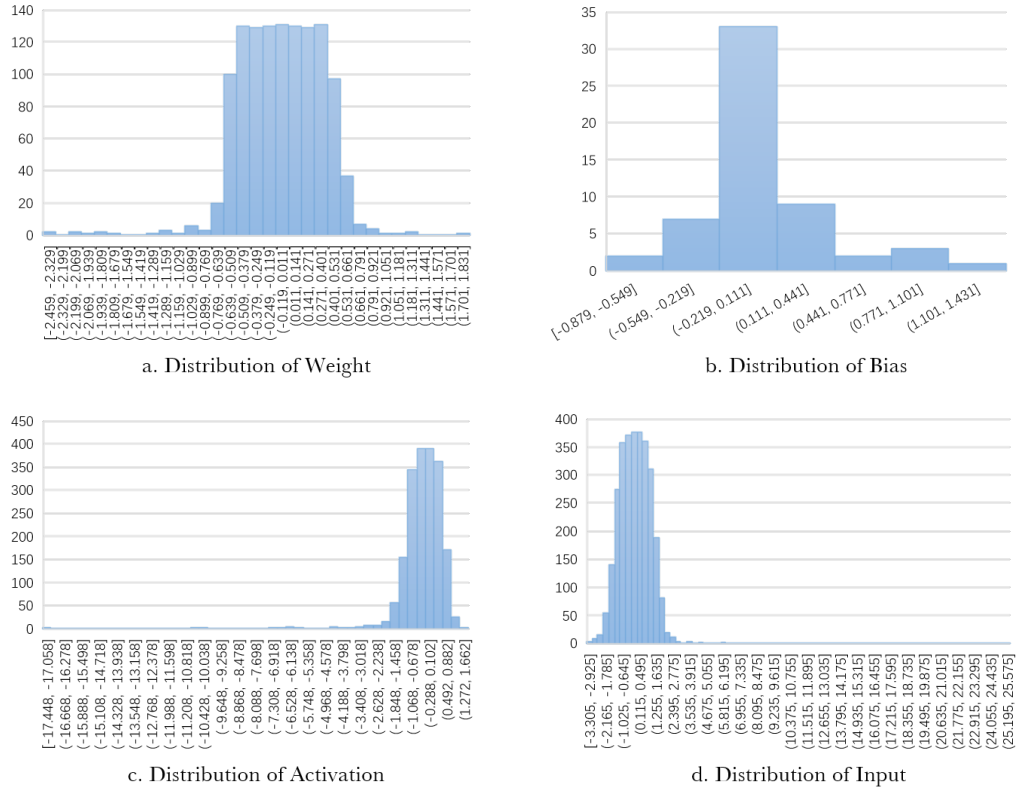

Figure 5. Histogram of value distribution: the horizontal axis indicates the range of values, and the vertical axis indicates the occurrences. We noticed that the values of the parameters were mainly concentrated in a very small interval, which was the basis of the quantization.

Table 7. Structure of the final network.

Layer	Input Dim.	Output Dim.	Kernel	Dilation	Stride	Ops(MAC)
Input	-	1024	-	-	-	-
Conv1	1024	1024	3	1	1	3072
Conv 2	1024	1024	3	2	1	3072
Conv 3	1024	1024	3	4	1	3072
Conv 4	1024	1024	3	1	1	3072
Conv 5	1024	1024	3	2	1	3072
Conv 6	1024	1024	3	4	1	3072
Conv 7	1024	1024	3	1	1	3072
Conv 8	1024	1024	3	2	1	3072
Conv 9	1024	1024	3	4	1	3072
Conv 10	1024	1024	3	1	1	3072
Conv 11	1024	1024	3	2	1	3072
Conv 12	1024	1024	3	4	1	3072
Conv 13	1024	1024	3	1	1	3072
Conv 14	1024	1024	3	2	1	3072
Conv 15	1024	1024	3	4	1	3072
Conv 16	1024	1024	3	1	1	3072
Conv 17	1024	1024	3	2	1	3072
Conv 18	1024	1024	3	4	1	3072
Conv 19	1024	1024	3	1	1	3072
Conv 20	1024	1024	3	2	1	3072
Conv 21	1024	1024	3	4	1	3072
Conv 22	1024	1024	3	1	1	3072
Conv 23	1024	1024	3	1	1	3072
Conv 24	1024	1024	3	1	1	3072
FC 25	1024	34	-	-	-	34,816

5. Hardware Implementation

5.1. Overall Architecture

Figure 6 shows the overall architecture of the accelerator designed in this paper. The FPGA accelerator consists of a controller, on-chip memory, input/output buffers, and several computation units. The controller is responsible for receiving commands from the host CPU and controlling the state of the accelerator. The on-chip memory consists of BRAM of different sizes for storing weights. The input/output buffer is composed of two 512-bit wide FIFOs (first-in, first-out) that buffer the data between the off-chip memory and the on-chip memory.

In most CNN accelerators, the size of feature maps between two layers often significantly overflows the size of the on-chip memory in the FPGA. This requires that the intermediate results be stored to off-chip memory, which takes up lots of time. However, the feature maps between the two dilated convolutional layers of our network is only 2 kilobytes at most and we do not need to cache all intermediate results with fully pipelined architecture. Therefore, our accelerator only accesses off-chip memory for obtaining inputs and returning outputs.

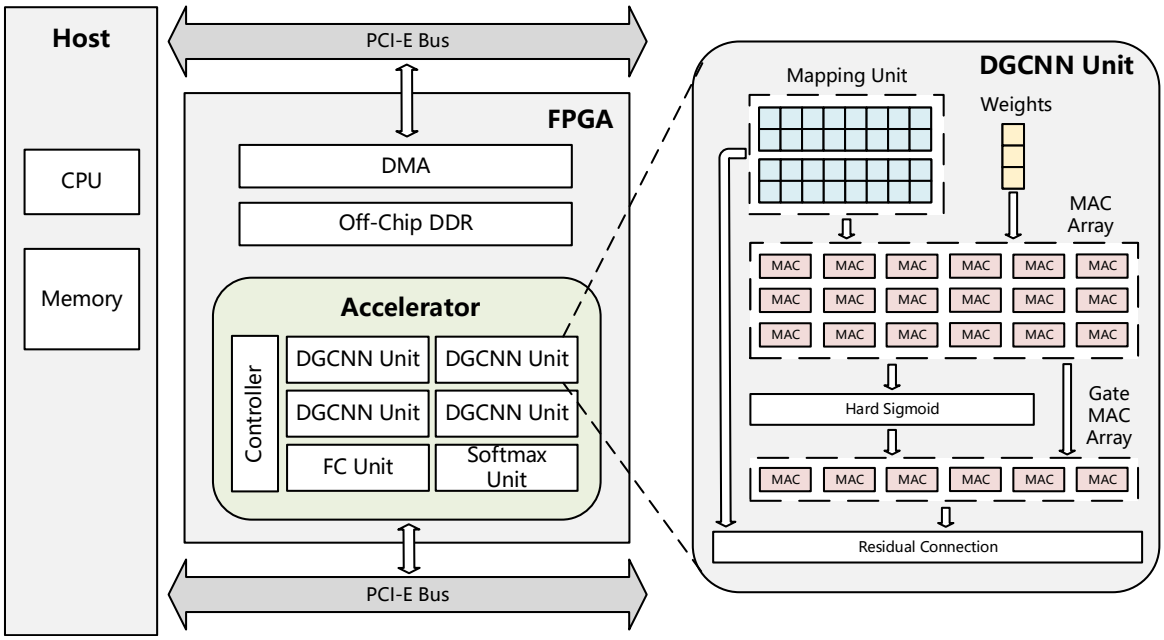

Figure 6. Overall architecture of our accelerator (**left**) and Dilate Gated Convolutional Neural Network (DGCNN) unit (**right**).

5.2. Compute Unit Architectures

Mapping Unit. The different dilations result in different requirements for loading in parallel. In order to meet the parallel computing requirement, we need to split and regroup the input data to make sure that the computing unit can access the necessary input data at the same time. The mapping unit mainly consists of mapping logic and a variable number of linebuffer. Suppose the input parallelism is p and the dilated rate is d, then the data needs to be split into $2 \times (d-1) + p$ banks. Figure 7 shows how the input data is partitioned when p is 4 and d is 2.

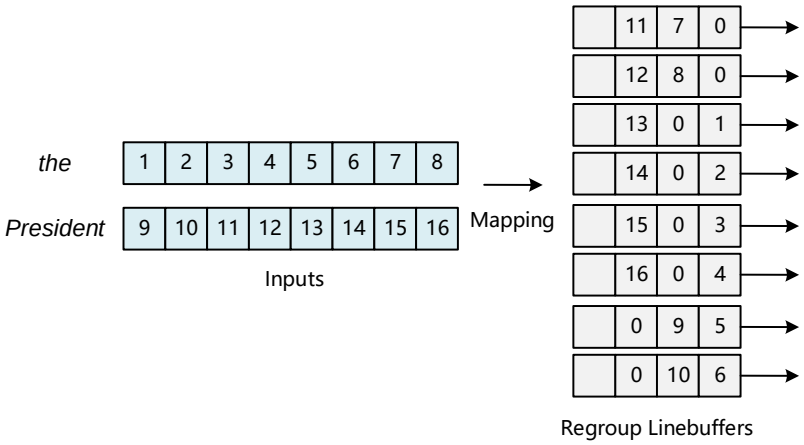

Figure 7. Example for input mapping strategy with $p = 4$ and $d = 2$.

DGCNN Unit. The DGCNN unit is the most critical part of the accelerator. As shown in Figure 6, the DGCNN unit consists of the mapping unit, input buffer, MAC (Multiply ACcumulate) array, CONV (CONVolution) buffer, and gate MAC array. Particularly, every weight of a dilated gated convolutional layer only involves three 8-bit numbers which can be stored in on-chip memory easily. Input buffer and CONV buffer are composed of several line buffers, which are used to cache the data in the pipeline. The MAC array contains $3 \times p$ Digital Signal Processing (DSP) slices, where p is the parallelism of the input data. The gate MAC array consists of p DSP slices for gate mechanism and addition logic for residual structure. The computational flow of DGCNN unit includes 5 stages.

1. Regroup the input by the mapping unit.
2. The MAC array computes the convolution, while the input buffer caches the inputs for the gate structure.
3. Get the sigmoid results of convolutional results by shift and addition operations.
4. The gate MAC array calculates the result of gated convolution.
5. Calculate the residual results.

FC Unit. The structure of the FC unit is comparatively simple. It consists of the input buffer, MAC array, and output buffer. Each cycle input buffer loads the output result of the previous layer and broadcasts it in the MAC array. The MAC array performs multiplication and accumulation calculations based on the control signal and updates the output buffer.

Softmax Unit. Softmax is often the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted classes. It is defined as Equation (5):

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{5}$$

Obviously, Softmax is as unsuitable for hardware computing as Sigmoid. The difference is that the purpose of the Softmax function is to find the most probable category, which corresponds to the maximum value of inputs. Therefore, we ignored the complex mathematical calculations and used a pipelined comparator with 34 inputs to get the Softmax result.

5.3. Layer Fusion

Unlike common CNNs, each convolutional layer of our DGCNN model has the same computational complexity and its throughput rate is exactly the same. Therefore, this paper constructed a fine-grained pipeline by fusing all network layers to reduce computational latency.

Intralayer Pipeline. For a convolutional layer, the fastest way to start outputting the results is to compute simultaneously within the same convolutional window. The ideal situation is to complete all

computations for a group of inputs at the same time. In this paper, this means that we need to calculate 3072 multiplications and additions at the same time. This not only requires a large number of multiplier, but also needs high memory bandwidth. It is obvious that computing in full parallel is extremely difficult on the hardware. In our model, the dilated convolution is a 1-D convolution. The data dependence of different layers is unidirectional. Therefore, it is important to focus on outputting the results as quickly as possible when computing in parallel in the layer. We fully unrolled the loop of the kernel and limit the unrolled between sliding windows to balance resource constraints and performance requirements.

Interlayer Pipeline. We designed a fine-grained interlayer pipeline to reduce the overall latency of the network. All layers of the network are fused to one layer. Take the example of the 3-layer network shown in Figure 8. The computation of the second layer will be started immediately when all the blue elements of the first layer are ready. Similarly, the computation of the third layer starts immediately after the second layer has computed the yellow elements. After the third layer finishes calculating, the results are saved instantly to reduce unnecessary memory usage. In brief, the latencies of the different layers are overlapped. Therefore the overall latency is much smaller than the sum of the latencies of all layers.

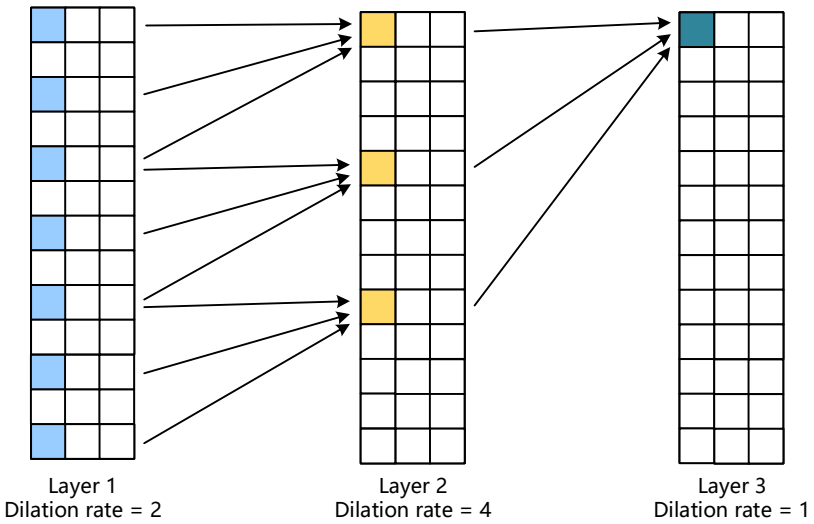

Figure 8. Data dependency of DGCNN.

Figure 9 shows the traditional pipeline and the layer-fused pipeline. It is clear that the layer-fused pipeline can significantly reduce the data processing latency.

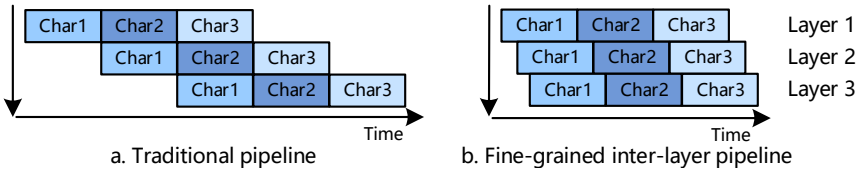

Figure 9. The proposed fine-grained interlayer pipeline for layer fusion: (**a**) in a traditional interlayer pipeline, subsequent computations require all of the previous layer's results. (**b**) The proposed fine-grained interlayer pipeline requires only a small number of results for starting computations.

Figure 10 shows the latency of each layer and the entire network. The overall latency is only 1.7 times greater than the single layer.

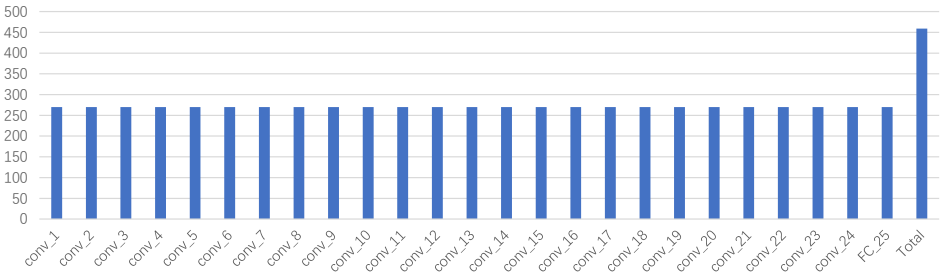

Figure 10. The latency of each layer and total network.

6. Experiments

6.1. Experimental Setup

The ACE 2005 corpus is based on real-world radio, news, and web blogs. It can reflect the performance of models in real-world scenarios. We used the same data split for training and testing as in previous studies [44]. Accuracy cannot properly measure the performance of the model with uneven class distribution, which is common in NLP tasks. Therefore, we used precision, recall, and F1-score to evaluate our event detection model. Precision is the ratio of correctly predicted positive labels to the total predicted positive labels. Recall is the ratio of correctly predicted positive labels to all labels except for NONE. We can tell that both precision and recall only measure the model in an isolated dimension. The F1-score takes both precision and recall into account and provides a better measure of model performance. It can be calculated by Equation (6):

$$\text{F1-score} = \frac{2 \times precision \times recall}{precision + recall} \tag{6}$$

A trigger is considered to be correct if its type and offset match the correct label. In most trainings, we set the learning rate to $1e-3$, the maximum sequence length to 128, and the batch size to 4 and used Adam as the gradient descent optimizer.

The accelerator is based on the Xilinx XCKU115 FPGA chip. It is written in Verilog, and all syntheses are from Xilinx Vivado 2018.3. The general purpose processing platform is based on the Intel Core i7-8700k CPU and the NVIDIA GTX 1080 GPU. The code leverages Keras and calls CUDA (Compute Unified Device Architecture) for GPU.

6.2. Evaluation of the Model

We chose previous works which are also based on the ACE 2005 corpus as the baseline for comparison. As shown in Table 8, our model significantly outperformed previous works before 2019. Our results also outperformed previous BERT-based works. Compared to the original EE-DGCNN model, our optimization strategy results in a nearly 12% point improvement in F1-score. Moreover, our model is more suitable for hardware acceleration than other works (e.g., Bi-LSTM [29] and NPN [23] (Nugget Proposal Networks)).

Table 8. Model performance comparison.

Model	Precision	Recall	F1-Score
DMCNN [22]	61.6	58.8	60.2
C-BiLSTM [7]	60	60.9	60.4
NPN [23]	60.9	69.3	64.8
TLNN [28]	64.5	71.5	67.8
Bi-LSTM+CRF [29]	66.4	76	70.9
BERT-CRF [44]	no report	no report	**83.2**
EE-DGCNN	90.5	60.8	72.7
Ours	96.1	**75.6**	**84.6**

Moreover, the precision of the DGCNN-based model is significantly higher than the recall. By analyzing the experiments and previous works, we believe that two causes resulted in this phenomenon. First, the BERT-based word representation can significantly improve precision with little effect on the recall. Balali et al. [46] obtained a 6.7% point increase in precision and only a 0.69% point increase in recall using the BERT-based word representation compared to the glove-based word representation with the same model. Second, DGCNN is a character-wise model. In contrast to the word-wise model, the character-wise model lacks information about the location of the characters. Meanwhile, our evaluation requires that a word be considered correctly classified only if all characters inside the word are classified correctly. This also affects the precision and recall [29].

6.3. Evaluation of the Accelerator

The resource utilization of our implementation is reported in Table 9. We can tell that our DGCNN accelerator is very small. All resource utilization percentages are less than 15%. This means that our proposed hardware architecture can be implemented on the various resource-limited platforms (e.g., embedding platforms).

We made comparisons with general purpose processing platforms. As shown in Table 10, we compared based on the Intel Core i7-8700k and NVIDIA GTX1080. Note that the power of the CPU is its thermal design power, the value of which is from [47], while the GPU power value was from the *nvidia-smi* program. FPGA power was reported by Xilinx Vivado 2018.3. As we can see, our accelerator is significantly superior to both CPU and GPU both in throughput and energy efficiency. Our throughput and energy efficiency are 17× and 158× higher than the GPU, respectively. We analyzed the reasons for the low performance of CPUs and GPUs. The reason could be caused by the fact that the deep learning framework was not yet optimized for the DGCNN structure, resulting in inefficient computation. This can be seen by the fact that the GPU with the thermal design power of 180 W [48] consumed only 66 W of power at full utilization.

Table 9. Resource utilization.

Resource	LUT	FF	BRAM	DSP
Utilization	42,146	52,765	309	619
Available	663,360	1,326,720	2160	5520
Utilization %	6.353413	3.977101	14.30556	11.21377

Table 10. Performance comparison between different platforms.

Platform	CPU	GPU	FPGA
Model	Intel Core i7-8700k	NVIDIA GTX1080	Xilinx XCKU115
Frequency	3.7 GHz	1607 MHz	200 MHz
Latency (us)	154.0	43.0	2.5
Power (W)	95.0	66.0	7.1
Perf. (kWord/s)	6.5	25.8	438.6
Throughput (GOPS)	1.4	5.6	95.2
Efficiency (GOPS/W)	0.01	0.1	13.4

7. Conclusions

In this paper, we analyzed the event detection task and proposed a hardware-friendly Chinese event detection model based on EE-DGCNN. We optimized the model by adjusting the dilation and by replacing the Sigmoid function to make it more suitable for hardware implementation. Additionally, we quantized the parameters and activations to further reduce hardware complexity and resource utilization. The model achieved the best F1-score on the Chinese ACE 2005 corpus. We further proposed an accelerator architecture and implemented it on a Xilinx XCKU115 FPGA. Our accelerator adopted a full pipelined architecture, which significantly reduces the latency by

combining interlayer and intralayer pipelines. Our experiments show that the accelerator achieved 95.2 GOP/s and 13.4 GOPS/W in performance and energy efficiency, which is 17 and 158 times higher than the GPU. To our knowledge, we are the first to propose an accelerator for natural language processing tasks. Future works should explore how to combine event detection and event arguments extraction and should extend benchmark results to other datasets (e.g., KBPEval2017).

Author Contributions: Conceptualization, Z.H., J.J., and L.Q.; methodology, Z.H., J.J., J.X., and Y.D.; software, Z.H., L.Q., and Z.K.; writing—original draft preparation, Z.H.; writing—review and editing, Z.H.; supervision, J.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Natural Science Foundation of China (61732018) and Pre-Research Foundation (31513010602-1).

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

Abbreviations

The following abbreviations are used in this manuscript:

1-D	One-Dimensional
2-D	Two-Dimensional
ACE	Automatic Content Extraction
BERT	Bidirectional Encoder Representations from Transformers
BRAM	Block Random Access Memory
CNN	Convolutional Neural Network
CONV	CONVolution
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DCNN	Dilate Convolutional Neural Network
DGCNN	Dilate Gated Convolutional Neural Network
DMCNN	Dynamic Multi-pooling Convolutional Neural Network
DNN	Deep Neural Network
DSP	Digital Signal Processing
ED	Event Detection
ELMo	Embeddings from Language Models
FC	Fully Connected
FIFO	first-in, first-out
FPGA	Field-programmable Gate Array
GOP	Giga Operation
GOPS	Giga Operations Per Second
GPT	Generative Pretrained Transformer
GPU	Graphics Processing Unit
IE	Information Extraction
LSTM	Long Short-Term Memory
MAC	Multiply ACcumulate
ML	Machine Learning
NLP	Natural Language Processing
NPN	Nugget Proposal Networks
PoS	Part-of-Speech
RNN	Recurrent Neural Networks
RoBERTa	Robustly optimized BERT approach
SQuAD	The Stanford Question Answering Dataset
TOP	Tera Operation

References

1. Ji, H.; Grishman, R. Refining event extraction through cross-document inference. In Proceedings of the ACL-08: Hlt, Columbus, OH, USA, 15–20 June 2008; pp. 254–262.
2. Tanev, H.; Piskorski, J.; Atkinson, M. Real-time news event extraction for global crisis monitoring. In *Proceedings of the International Conference on Application of Natural Language to Information Systems, London, United Kingdom, 24–27 June 2008*; Springer: Berlin/Heidelberg, Germany; pp. 207–218.
3. Liao, S.; Grishman, R. Using document level cross-event inference to improve event extraction. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Uppsala, Sweden, 11–16 July 2010; pp. 789–797.
4. Hogenboom, F.; Frasincar, F.; Kaymak, U.; De Jong, F. An overview of event extraction from text. In Proceedings of the DeRiVE@ ISWC, Bonn, Germany, 23–27 October 2011; pp. 48–57.
5. Li, D.; Huang, L.; Ji, H.; Han, J. Biomedical event extraction based on knowledge-driven tree-LSTM. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapoils, MN, USA, 2–7 June 2019; pp. 1421–1430.
6. Li, L.; Zheng, J.; Wan, J. Dynamic extended tree conditioned lstm-based biomedical event extraction. *Int. J. Data Min. Bioinform.* **2017**, *17*, 266–278.
7. Zeng, Y.; Yang, H.; Feng, Y.; Wang, Z.; Zhao, D. A convolution BiLSTM neural network model for Chinese event extraction. In *Natural Language Understanding and Intelligent Applications*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 275–287.
8. Wang, Y.; Wang, J.; Lin, H.; Zhang, S.; Li, L. Biomedical event trigger detection based on bidirectional lstm and crf. In Proceedings of the 2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Kansas City, MO, USA, 13–16 November 2017; pp. 445–450.
9. Sha, L.; Qian, F.; Chang, B.; Sui, Z. Jointly Extracting Event Triggers and Arguments by Dependency-Bridge RNN and Tensor-Based Argument Interaction. In Proceedings of the AAAI 2018, New Orleans, LA, USA, 2–7 February 2018; pp. 5916–5923.
10. Yu, X.; Rong, W.; Liu, J.; Zhou, D.; Ouyang, Y.; Xiong, Z. LSTM-Based End-to-End Framework for Biomedical Event Extraction. In *IEEE/ACM Transactions on Computational Biology and Bioinformatics*; IEEE: New York, NY, USA, 2019.
11. Björne, J.; Salakoski, T. Biomedical event extraction using convolutional neural networks and dependency parsing. In Proceedings of the BioNLP 2018 Workshop, Melbourne, Australia, 19 July 2018; pp. 98–108.
12. Nguyen, T.H.; Grishman, R. Event detection and domain adaptation with convolutional neural networks. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), Beijing, China, 26–31 July 2015; pp. 365–371.
13. Nguyen, T.H.; Grishman, R. Modeling skip-grams for event detection with convolutional neural networks. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, TX, USA, 1–5 November 2016; pp. 886–891.
14. Nguyen, T.H.; Cho, K.; Grishman, R. Joint event extraction via recurrent neural networks. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego, CA, USA, 12–17 June 2016; pp. 300–309.
15. Ghaeini, R.; Fern, X.Z.; Huang, L.; Tadepalli, P. Event nugget detection with forward-backward recurrent neural networks. *arXiv* **2018**, arXiv:1802.05672.
16. Ciresan, D.C.; Meier, U.; Masci, J.; Gambardella, L.M.; Schmidhuber, J. Flexible, high performance convolutional neural networks for image classification. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, Barcelona, Spain, 16–22 July 2011.
17. Rawat, W.; Wang, Z. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Comput.* **2017**, *29*, 2352–2449.
18. Karpathy, A.; Toderici, G.; Shetty, S.; Leung, T.; Sukthankar, R.; Fei-Fei, L. Large-scale video classification with convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 1725–1732.
19. Kappeler, A.; Yoo, S.; Dai, Q.; Katsaggelos, A.K. Video super-resolution with convolutional neural networks. *IEEE Trans. Comput. Imaging* **2016**, *2*, 109–122.

20. Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; Kuksa, P. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.* **2011**, *12*, 2493–2537.
21. Yandi, X.; Yang, L. Chinese event extraction using deep neural network with word embedding. *Comput. Lang* **2016**, *1*, 1–6.
22. Chen, Y.; Xu, L.; Liu, K.; Zeng, D.; Zhao, J. Event extraction via dynamic multi-pooling convolutional neural networks. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Beijing, China, 26–31 July 2015; pp. 167–176.
23. Lin, H.; Lu, Y.; Han, X.; Sun, L. Nugget proposal networks for chinese event detection. *arXiv* **2018**, arXiv:1805.00249.
24. Bahdanau, D.; Cho, K.; Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv* **2014**, arXiv:1409.0473.
25. Shao, Y.; Hardmeier, C.; Tiedemann, J.; Nivre, J. Character-based joint segmentation and POS tagging for Chinese using bidirectional RNN-CRF. *arXiv* **2017**, arXiv:1704.01314.
26. Chiu, J.P.; Nichols, E. Named entity recognition with bidirectional LSTM-CNNs. *Trans. Assoc. Comput. Linguist.* **2016**, *4*, 357–370.
27. Tai, K.S.; Socher, R.; Manning, C.D. Improved semantic representations from tree-structured long short-term memory networks. *arXiv* **2015**, arXiv:1503.00075.
28. Ding, N.; Li, Z.; Liu, Z.; Zheng, H.; Lin, Z. Event detection with trigger-aware lattice neural network. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, 3–7 November 2019; pp. 347–356.
29. Xi, X.; Zhang, T.; Ye, W.; Zhang, J.; Xie, R.; Zhang, S. A Hybrid Character Representation for Chinese Event Detection. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; pp. 1–8.
30. Chang, J.-W.; Kang, S.-J. Optimizing fpga-based convolutional neural networks accelerator for image super-resolution. In Proceedings of the 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), Jeju, Korea, 22–25 January 2018; pp. 343–348.
31. Zhao, M.; Hu, C.; Wei, F.; Wang, K.; Wang, C.; Jiang, Y. Real-time underwater image recognition with FPGA embedded system for convolutional neural network. *Sensors* **2019**, *19*, 350.
32. Zhang, X.; Liu, X.; Ramachandran, 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; pp. 1–4.
33. Bettoni, M.; Urgese, G.; Kobayashi, Y.; Macii, E.; Acquaviva, A. A convolutional neural network fully implemented on fpga for embedded platforms. In Proceedings of the 2017 New Generation of CAS (NGCAS), Genova, Italy, 6–9 September 2017; pp. 49–52.
34. Kan, Z.; Qiao, L.; Yang, S.; Liu, F.; Huang, F. Event Arguments Extraction via Dilate Gated Convolutional Neural Network with Enhanced Local Features. *arXiv* **2020**, arXiv:2006.01854.
35. ACE 2005 Multilingual Training Corpus. Available online: https://catalog.ldc.upenn.edu/LDC2006T06 (accessed on 30 August 2020).
36. Yu, F.; Koltun, V. Multi-scale context aggregation by dilated convolutions. *arXiv* **2015**, arXiv:1511.07122.
37. Yin, W.; Kann, K.; Yu, M.; Schütze, H. Comparative study of cnn and rnn for natural language processing. *arXiv* **2017**, arXiv:1702.01923.
38. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. *arXiv* **2013**, arXiv:1301.3781.
39. Peters, M.E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep contextualized word representations. *arXiv* **2018**, arXiv:1802.05365.
40. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. Improving Language Understanding by Generative Pre-Training. 2018. Available online: https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf (accessed on 10 October 2020).
41. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.

42. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv* **2019**, arXiv:1907.11692.
43. Gehring, J.; Auli, M.; Grangier, D.; Yarats, D.; Dauphin, Y.N. Convolutional sequence to sequence learning. *arXiv* **2017**, arXiv:1705.03122.
44. M'hamdi, M.; Freedman, M.; May, J. Contextualized Cross-Lingual Event Trigger Extraction with Minimal Resources. In Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL), Hong Kong, China, 3–4 November 2019; pp. 656–665.
45. Hou, L.; Zhang, R.; Kwok, J.T. Analysis of quantized models. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
46. Balali, A.; Asadpour, M.; Campos, R.; Jatowt, A. Joint Event Extraction along Shortest Dependency Paths using Graph Convolutional Networks. *arXiv* **2020**, arXiv:2003.08615.
47. Intel® Core™ i7-8700K Processor (12M Cache, up to 4.70 GHz) Product Specifications. Available online: https://ark.intel.com/content/www/us/en/ark/products/126684/intel-core-i7-8700k-processor-12m-cach\e-up-to-4-70-ghz.html (accessed on 30 August 2020).
48. GeForce GTX 1080 Graphics Cards | NVIDIA GeForce. Available online: https://www.nvidia.com/en-sg/geforce/products/10series/geforce-gtx-1080/ (accessed on 30 August 2020).