**An Energy-efficient Convolutionalal Neural Network Accelerator for Speech Classification Based on FPGA and Quantization**

**Dong Wen1, Jingfei Jiang1,\*, Yong Dou1, Jinwei Xu1, Tao Xiao1**

**Abstract**

CNN (Deep convolutional neural network), which is widely applied in image tasks, can also achieve excellent performance in acoustic tasks. However, activation data in convolutional neural network is usually indicated in floating format, which is both time-consuming and power-consuming when be computed. Quantization method can turn activation data into fixed-point, replacing floating computing into faster and more energy-saving fixed-point computing. Based on this method, this article proposes a design space searching method to quantize a binary weight neural network. A specific accelerator is built on FPGA platform, which owns layer-by-layer pipeline design, higher throughput and energy-efficiency compared with CPU and other hardware platforms.

**[[1]](#footnote-1)**

**Keywords**: FPGA; quantization; reconfigurable computing; energy-efficiency; speech classification

1. **Introduction**

Speech classification is a typical information analyzing task, which is widely used in military and speech controlling. Since Deng, Yu et al. introduced RNNs (Recurrent Neural Networks) acoustic models into speech recognition and speech classification, RNN has reached series of excellent performance in this area (Geoffrey H, et al, 2012; Wan H, et al 2019). However, deep neural networks based on RNN are hard to be trained and parallelized due to complex structure and recurrent computation. When applied in actual missions, RNN models usually demand high performance GPU and CPU. Such hardware platforms have up to hundreds of watt power consumption, which cannot meet requirements for energy-sensitive circumstance. On the contrast, CNN has been found to get excellent performance on acoustic model (Tom S, et al, 2016; Muckenhirn H, et al, 2018; Palaz D, et al, 2015). Audio files can be transferred into feature maps or feature matrixes by wave-filtering algorithms (such as Mel Frequency Cepstral Coefficients algorithm) (Pakyurek M, et al, 2020), then these maps are input into acoustic CNN models, just like input images for computer vision models. Because convolution computation is easier to be parallelized and accelerated than recurrent computation, CNNs with tiny 3x3 or 5x5 convolution kernels are able to be trained and inferenced faster than RNNs. This advantage makes it possible to accelerate an acoustic CNN model on specific power-efficiency hardware platforms.

Gradient descent algorithm (Jyrki Kivinen M K W, et al, 1997), which is sensitive to numerical fluctuation (Perkins S, et al, 2003), is widely applied to train deep neural networks (DNN). To pursue best DNN performance, it is necessary to store data in full-precision format during training process. However, floating data format needs longer word-length to store and more circuit parts to compute, leading to more energy consumption and bigger circuit designing area (Liu S, et al, 2011). Also, the computing complexity of floating-point data makes it difficult to reduce computing cycle counts. Although floating computation can keep precision well, it has become the bottleneck of power-efficiency high performance computing.

Fortunately, some works (Han S, et al, 2016; Dundar G, et al, 1995) have proven that floating-point data is unnecessary to CNNs’ forwarding tasks, low precision computing can achieve similar performance as well. These works present quantization methods, turning weight and activation data into fixed-point data, integer data or even binary data with little accuracy loss. Based on kinds of quantized CNN models, there comes BNN (Binary Neural Network) accelerators (Guo P, et al, 2019; Liang S, et al, 2018; Conti F, et al, 2018) and GPUs supporting 8 bits integer data (Michael D, Ashish K, David R. Nvidia’s Xavier et al, 2018; Nvidia. 2018) , etc. These designs reduce power consumption greatly and have up to hundreds of speedup ratios compared with CPU platforms. It turns out that hardware with corresponding quantized CNN models can achieve excellent computing performance as well as high energy efficiency.

Compared with CPU and GPU, ASIC and FPGA are more suitable to accelerate a specific task. These reconfigurable platforms can be customized by setting pipeline and expanding parallelism degree, lowering power consumption and raising computing performance. Although ASIC owns huge advantages over FPGA on power and speed, expensive designing and manufacturing cost limits it’s general application. On the contrast, FPGA keeps a good balance between performance, power, flexibility and expense due to programmable feature and mature industry design. Now, FPGA has been widely used in cloud computing and intelligence computing by Microsoft, Amazon and Alibaba (Gwennap L, et al, 2017; Turan F, et al, 2020), becoming an important part of high- performance computing.

The speech classification model which focuses on specific speech instructions or acoustics signal, is a basic component of intelligent scenario analysis in both cloud and edge end. Such applying circumstance needs a low-power but high-performance computing platform especially. Typical deep convolutional neural networks can do coarse speech classification work well. However, there still exists some space to accelerate CNN model and reduce computing platform’s energy consumption by quantization and customized hardware design. To implement this power-efficient speech classification computation platform, we choose a typical CNN-based speech classification model where weight value is +1 or -1 and activation data is in full precision floating-data format (Bo L, et al, 2018). We design an accelerator based on Xilinx XCKU-115 FPGA platform and run this BWN (Binary Weight Network) model. Compared with state-of-the-art CPU platforms, our accelerator achieves 18-300x throughput speed up ratio and high energy efficiency. The main contributions of this work are as follows:

1. We turn floating feature data into fixed-point data in a rather common method. However, as to super parameter like batch-normalization arguments, we take a special bitwise allocation method and apply design space exploration methods to find best quantization scheme. We also sperate intermediate result from feature data and take different quantization settings. Based on usual quantization method, we take these operations to help to decrease accuracy loss caused by quantization and achieve a closed model prediction accuracy compared with non-quantized model.

2. We design a novel layer-by-layer pipeline structure on our multi-PE (Process Element) BWN (Binary Weight Network) accelerator, which has shared weight storage, balanced pipeline partition and bitwise expansion. The performance, power consumption and energy efficiency of this accelerator are discussed in Section 5.

3. The target speech classification model is tested on CPU platforms to get performance baseline. Compared with these test results, our design has absolute advantage on performance per watt and throughput.

**2. Neural Network Inferencing Quantization**

When training a deep neural network, researchers usually choose full-precision data format to ensure best model accuracy. However, in inference task, these parameters will not be changed and therefore we can prune them in an offline method. (Bo L, et al, 2018) raises an algorithm to compress floating-point DNN parameters into binary data, which is consisted of +1 and -1. Compared with common DNN with floating-point weight and activation, this compression method not only sharply reduces parameter storage, but also replaces multiplication and division with add and minus. Less storage space and multiplication mean less memory-consuming energy and less computing cycles, leading to faster lower power consumption and faster working speed.

(Matthieu C, et al, 2016) brings out a method to turn activation into binary format. Unlike parameter in neural networks, activation data fluctuates numerically with different input (e.g., input image or input audio feature map). Although (Matthieu C, et al, 2016) still keeps a good model accuracy on very deep CNNs like VGGNet, great numerical precision loss would be brought out binary activation data, which may cause vital influence on some small-size CNNs (Jacob B, et al, 2017; Xu Y, et al, 2018). In this situation, turning floating data into fixed-point format can keep a good balance between computing performance and model accuracy: fixed-point data computation needs less computing cycles compared with floating data, and fixed-point can adapt to data’s numerical distribution by flexible allocation of integer bitwise and decimal bitwise.

When integer part is allocated with more bitwise, it can expand data-indication range; and when we give decimal part longer word-length, it can achieve better numerical precision correspondingly. For example, given a decimal number 10.06. With four-bit integer part and five-bit decimal part, this number can be indicated as 1010.00001 in binary format, which is 10.0625 when again translated into decimal number. Numerical precision loss here is generated by binary data’s discrete indicating method. With same integer part length and ten bit allocated to decimal part, the number 10.06 can then be reset as 1010.0000011111, which is 10.060546875 when translated into original format. This example shows that quantized can keep better numerical precision with longer decimal bit length. However, increasing the length of fixed-point data format will add cycle counts to computation or occupy more hardware’s resource on FPGA. Taking speed and computation precision into account, it is important to find the relatively best data format.

**3. Speech Classification Model**

**3.1 Model Architecture and Weight Binarization**

This CNN-based speech recognition model is trained on Tensorflow speech command dataset. It can recognize the six sort of short speech segment “up”, “down”, “yes”, “right”, “left” and “unknown words”. This model first uses MFCC algorithm turning an audio file into a floating format tensor, whose dimension is 20x49x1. Then this tensor will be sent into a convolutional neural network, which is consisted of two convolutional layers, three full-connected layers and binary weight parameters. The detail information of model architecture is shown in Fig. 1. All convolutional kernel size is 3 and convolutional stride is 1. There is no padding and expansion operation in this network, which is convenient for us to accelerate. To be noticed that activation is still in float format at this stage. Via softmax function, this model outputs the possibility of six type of labels.

After model parameters being fixed in the training process, we can transfer float weight value into (-1, 1). Fig. 2 shows how we processing weight data. We assume the distribution of primitive parameter is normal distribution, the numerical distributing range is then modified by tanh function and a series of scale methods. Finally, all parameters are discretized to -1 or +1. This stage’s BWN model (activation data is still in floating format) can provide the accuracy no less than 85%.

**Fig. 1** Convolutional Neural Network Architecture



**Fig. 2** Weight Binarization Process

**3.2 Quantizing Feature Data and intermediate Results**

Once we change floating input feature into fixed-point format, intermediate result, activation and other hyper-parameters (such as bias and batch-normalization parameter) will be in fixed-point format naturally. The data computed in each layer can be divided to two parts: intermediate results after MAC (Multiply-Accumulate) operations and the batch-normalized output data. The batch-normalized output data is feature data which will be transferred to next layer of neural network. In some cases, absolute value and variance value of intermediate results can be huge, in another word, the data distribution of intermediate result is rather fluctuant, which brings difficulty to quantization. These intermediate results will then be normalized, where both absolute value and variance are narrowed down. Unlike intermediate result data, the feature data follows normal distribution and easy to be quantized. To turn intermediate result and batch-normalized data into fixed-point format, the approximate data range of these two kinds of data needs to be determined first for numerical distribution range deciding data’s integer part bitwise. Intermediate result’s quantization thus demands longer integer indication for fluctuant data distribution.

On some IDEs (integrated development environments) running on CPU platforms such as Matlab, their quantization functions libraries usually are not the “real” quantization operations libraries. The intermediate results’ bit-width and precision loss cannot be reflected by such libraries for CPU platforms cannot handle fixed operations well. When running quantization operations, CPU will do floating computation first and then convert result into fixed-point format. However, on FPGA platforms, hardware design and resource utilization will be influenced by intermediate results because they usually demand for longer word length. As a result, FPGA has to use more LUT to complete such fixed-point computation and more register to store intermediate result.

To quantize feature data, we usually use eight bits for integer part (one for sign and seven for indicating value) and eight bits for decimal part in saturation mode, some layers’ feature data may need more decimal bits to increase indication precision. But it is obvious that this quantization setting is not suitable for intermediate results whose absolute value can go far beyond 127, we need to find a quantization format for intermediate results separately. Once we determine the absolute value’s range and corresponding integer bits for intermediate result, it is necessary to decrease decimal parts to reduce resource utilization. We try several bitwise schemes with design space search method and the experiment result is discussed in section 5.1.

**3.3 Quantizing Batch-normalization’s Parameters**

Batch-normalization operation, which improves data distribution greatly among neural network layers, has been proven to be vital to model’s final prediction accuracy (Santurkar S, et al, 2018; Liu M, et al, 2017). However, the multiplication and division in this process not only depend on intermediate results, but also are sensitive to parameters’ numerical precision (Giri E P, et al, 2016). The experiment in section 5.1 shows the decimal bitwise of normalization parameters, which reflects numerical precision, influence model’s prediction performance greatly.

Despite taking occupy of more hardware resource, allocating more decimal bits for batch-normalization parameters is proven to be important by experiment result. Considering the module size of batch-normalization is in small scale, our FPGA platform affords raising the computing precision in this process.

Based on original quantization idea, we separately raise three quantization formats for feature data, intermediate results and normalization parameters. We focus on keeping balance between hardware performance and numerical computing precision and apply search method to find best solution for intermediate results’ quantization. In the Experiment section, we will discuss detail quantization experiment results.

**4. Accelerator Architecture**

The target model is small, shallow and it’s weight is binarized. Thus, the main methods we focus are setting shared parameters storage on chip, accelerating neural network layer by layer and designing pipeline between layers. The general architecture of the accelerator is shown is Fig.3. Our accelerator contacts with DDR via input-data FIFO, output-data FIFO and series of address signal wires. Inside each process element, there comes different modules that mapped to different neural network’s layers. We can deploy layer-by-layer pipeline acceleration naturally by this design method. Also, shared parameter storage element is settled between each PE, reducing resource utilization on FPGA.



**Fig.3** Hardware Architecture

**4.1 Parameter Storage**

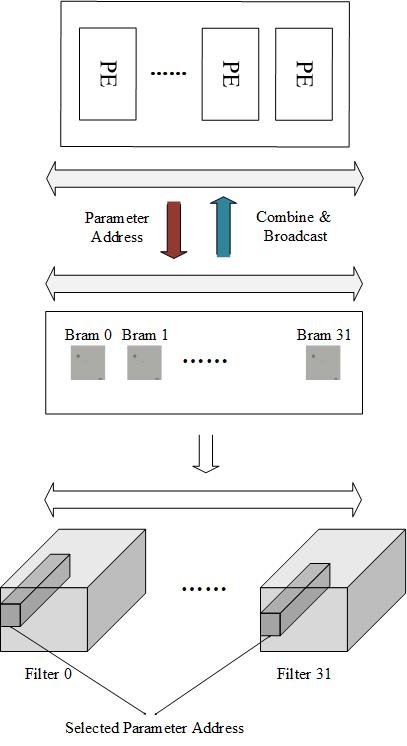
Unlike some neural network accelerators having to use DRAM instead of on-chip memory to store parameters (Cheung K, et al, 2012; Alessandro A, et al, 2018; Chen T, et al, 2015), our accelerator can store all parameters on-chip due to small-sized network architecture. With all parameters being stored on chip, we can save time-cost caused by communication with DRAM. The detail information of parameters is shown in Table. 1.

FC-1 layer occupies most of the parameter size, while other layers’ data is rather tiny and can be directly stored on chip. Considering the scale of FC-1 parameter, it is natural to share them between several PEs. In order to simplify the design, we set all PEs working synchronously and fetching exactly the same pre-trained data at FC-1 computing step. Shared memory structure is consisted of 32 block memory generators, each for one kernel in FC-1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layer | Filter | Kernel | Parameter | Parameter Size |
| conv1 | 32 | 3\*3\*1 | 288 | 36B |
| conv2 | 32 | 3\*3\*32 | 9216 | 1152B |
| fc1 | 32 | 16\*45\*32 | 737280 | 92160B |
| fc2 | — | — | 1024 | 128B |
| fc3 | — | — | 192 | 24B |

**Table.1 The Detail Information of Parameters**

The design detail of shared storage is demonstrated in Fig.4. PEs send target parameters’ addresses to shared storage structure. These addresses are line addresses of BRAM blocks so BRAM blocks can access data with no delay. Parameters are stored along the third dimension in BRAM, which is consistent to the input data’s organization method of FC-1 layer. Shared-parameter data is sent to all PEs by broadcasting.



**Fig.4** FC-1’s Parameter Sharing Structure

As to other layers’ binary weight, we store them directly inside modules that mapped to corresponding layers. Similar to the shared parameter structure, each BRAM block is responsible for one filter. When the vector processing unit in Fig.5 starts computing, BRAM blocks directly send related parameters to the unit and these binary data then compute with activation. The vector unit will either keep original activation value or reverse it due to the input binary weight (indicating +1 or -1), each channel will have 32 temporary results. All temporary data will then be put through parallel adder tree to compute for results, each channel will get one valid data every computation and 32 valid data for 32 channels.

Prior batch-normalization parameters also need to be storage on chip. To pursue quantization accuracy, these data bitwise varies. Therefore, it is better to storage batch-normalization parameter in flexible register.

**4.2 Bitwise Expansion**

Inside one convolutional or full-connected layer, the input fixed-point feature data is usually under normal distribution and varies in relatively small range. However, after MAC, the data’s variance can be huge and irregular (Wei Z, et al, 2017). Batch-normalization relies on these intermediate results’ mean and variance to improve data distribution, which is vital to final accuracy. If we simply apply the same data format as the input feature to intermediate results data in batch-normalization step, it will lead unnecessary loss to final prediction.

To handle this situation, we introduce bitwise expansion method which gives extra decimal bitwise to both feature data and normalization parameter when computing. After DSP outputting multiply results, these extra decimal bits will be cut and the data will return to original input format.

The batch-normalization operations can be divided into four steps: subtracting mean value, multiplying offset, dividing the square root of variance (which is substituted by multiplying the reciprocal of variance’s square root on our hardware design) and adding bias. As the Fig.6 shows, we implement these four parts of normalization in a pipeline way. The negative impact of multiply computation on FPGA is reduced in this way.



**Fig.5** Vector Processing Unit



**Fig.6** Batch-Normalization Unit

Similarly, intermediate results of MAC process have same needs for bitwise expansion. The different point is that MAC results usually need more bitwise in integer part instead of decimal number. We apply expansion to both normalization step and accumulator in adder tree of MAC, it turns out that this method ensures computing accuracy on hardware.

**4.3 Layer-by-layer Pipeline**

In deep convolutional neural networks such as VGG-16 and AlexNet (Simonyan K, et al, 2014; Krizhevsky A. et al, 2012), it is difficult to keep this balance due that as networks going deeper, deep layers will demand layers ahead to generate feature result at a faster speed, which is beyond current hardware’s computing power limit.

The target neural network is shallow and tiny, so it is possible to keep balance between different layers. The key point of layer-by-layer pipeline’s implementation is to balance running periods between different function layers in neural network. In convolutional neural networks, computation involves convolutional kernels and feature’s sub-map on specific position is the basic computing model. This operation is time-costing and needs large amount of circuit resource, so it is difficult to design balanced pipeline on resource-limited FPGA if choosing convolutional kernel’s computation as the basic grid of our pipeline. To decrease the waiting time and the starting cost of pipeline, we use the vector data along the channel direction in Fig.7 as our basic data operation unit.

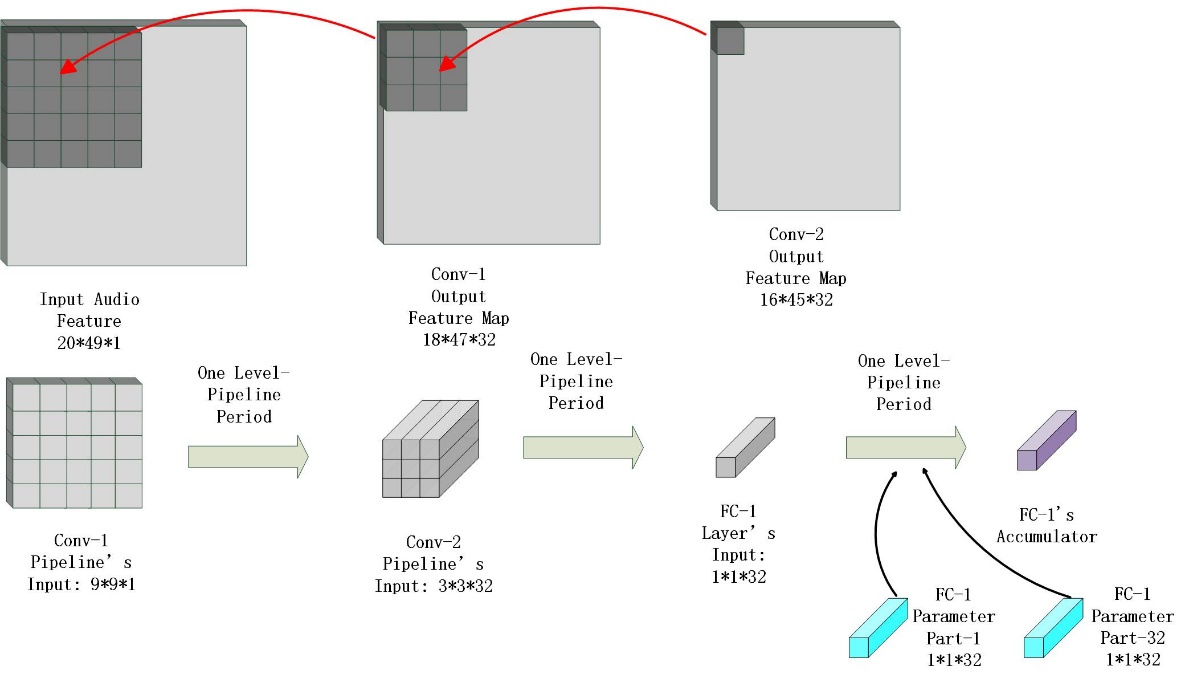
Fig.7 shows the cost of keeping balance between each network layer. To produce one unit of result in Conv-2 layers (which can just meets the need of starting full-connect layer’s pipeline) , this neural network has to compute one 3x3 slide window on Conv-1’s output map and compute nine 3x3 slide windows from a 5x5 area on input audio feature. We expand the scale of Conv-1’s computing array to make it can generate nine 32x1 vectors in one macro pipeline period by adjusting to this computation pattern. These vectors will be sent to Conv-2 and generate one 32x1 vector for FC-1 layer’s input in next period. In this way, Conv-1 layer, Conv-2 layer and FC-1 layer of the neural network can work in a pipeline mode, which in practice consists the “layer-by-layer” pipeline design.

In our neural network model, FC-1 layer occupies huge size of parameter and relatively small size of computation. More importantly, FC-1 layer starts after convolutional layers finishing computing corresponding data. It brings unbearable cost if FC-1 layer will wait until Conv-2’s computation is done, for it demands lots of on-chip storage resource to keep intermediate result and increase pipeline’s waiting time. To eliminate the bottleneck of whole pipeline, we set the arriving of channel direction’s vector data as FC-1layer’s starting condition. In each macro-period, FC-1 layer will get one vector (one load data) as input, fetch 32 groups of parameters from shared storage and compute them in one macro-period as well. The intermediate results only need one register which also functions as an accumulator. With the method shown in Fig.7, we finally get a balanced and high-performance layer-by-layer pipeline design.

**5. Experiment and Discussion**

**5.1 Quantized Model’s Performance**

We use Matlab-2018a’s quantizer function to turn feature and batch-normalization parameters into fixed-point format with saturate mode. In order to find the best bitwise allocation scheme, we conduct a couple of quantization experiments and compare their accuracy performance.



**Fig. 7** Balancing Conv-1 and Conv-2

To run neural networks on our specific hardware platform, the data format must fit the hardware design code. We set the bitwise of all data must range in 16 bits to 32 bits, which is hardware-friendly to our FPGA platform, as the boundary condition for design space searching. During actual experiments, we determine the approximate distribution range of feature data, intermediate results and normalization parameters, it shows that the integer bitwise of intermediate results usually needs 8~12 bits while normalization parameter needs 20~21 bits for integer part. This result can help us to determine the upper bound of decimal bitwise. In deep neural networks, activation and intermediate results are relatively robust to numerical precision, so we do not have to devote too much work on these data’s decimal bitwise. On the other hand, normalization parameters need more data format accuracy than intermediate results, so in principle, we give them decimal bitwise no less than intermediate result.

**Fig.8** Accuracy Experiment Result

The horizontal axis (IRDB, NRDB) means combination of Intermediate result Decimal Bitwise and Normalization Result (Feature Data) Decimal Bitwise.

To find best quantization bitwise allocation scheme, we conduct several experiments under the condition discussed ahead and experiments’ result is illustrated in Fig.8 and Fig.9. In Fig.8, vertical axis shows the corresponding model’s prediction accuracy with different combination of IRDB (Intermediate Result Decimal Bitwise) and NRDB (Normalization Result Decimal Bitwise). It turns out when middle data applies 8-bit decimal bitwise and normalization parameter uses 9-bit decimal bitwise, this model will achieve best accuracy. We also implement the experiment that intermediate result owns more decimal length than normalization data’s which violates our searching principle, the final result supports our idea effectively.

It is also noticeable that when (IRDB, NRDB) is over (8, 9), model’s final accuracy does not rise with better intermediate results and feature data’s indication precision. Some works have proven that neural networks have numerical robustness in low-precision data format (Zeng X, et al; Cheng G, et al), this feature is widely used by deep neural networks’ quantization and can keep model’s original prediction performance well. On the other hand, final accuracy can be improved by better indication precision at initial stage of experiment. We consider that the point (IRDB, NRDB) equaling (8, 9) lies in a balance condition where both robustness and numerical precision can function well. When decimal bits are increased and model is over this balancing point, model’s robustness is weakened by better data indication performance and this boost on numerical precision cannot cover the loss of robustness, which we consider is the reason why model’s prediction effect cannot be enhanced in last couples of experiments.

The experiment results on normalization parameters' quantization scheme are demonstrated in Fig.9. We set (IRDB, NRDB) as (8, 9) and keep other variables unchanged. It is clearly shown that model’s prediction accuracy is improved by better normalization parameters indication precision. However, the word length of these parameters cannot exceed the upper boundary 32 bits, so it is suitable to allocate 12 decimal bits for batch-normalization parameters. There is one thing need to be noticed, hardware cannot handle division operation as easy as Matlab code. Thus, we turn variance’s division in batch-normalization into reciprocal multiplication and expand decimal bit-width to ensure accuracy.

We run our non-quantization version of Matlab code on Intel core i7-8700K with and without multi-thread accelerating library. We divide whole program into several function segments and test their running time. Table.2 shows that when ignore the MFCC segment, the second convolutional layer is performance bottleneck on CPU platforms and has vital effect to the model, which is corresponded to the largest computing scale of Conv-2 layer. Data in Table.2a also illustrates that although parallelism function library is used and running time is reduced sharply, Conv-2 is still the most time-costing layer on CPU.

Quantization version of code runs 9.85% slower than non-quantization version, detail is shown in Table.2b. As discussed in section 3.2, although we apply quantization functions and fixed-point computing in the code, it is still executed in floating mode on CPU for CPU platforms cannot process fixed-point data well. This procedure brings extra work converting data between floating format and fixed-point format, which is responsible for worse performance of quantized code. For the similar reason, different quantization formats will not make noticeable impact on running time on CPU platforms. To truly reflect our accelerator’s performance, we do not choose quantization version’s running time but the non-quantization version’s as our performance baseline.

|  |  |  |
| --- | --- | --- |
| 8700K without Parallelism | | |
| Function Segment | Time (Seconds) | Ration in Total Time |
| Data Loading & Pre-process | 9.413601 | 7.40% |
| Conv-1 | 3.084229 | 2.43% |
| Conv-2 | 114.582 | 90.14% |
| FC-1 | 0.023377 | <1% |
| FC-2 | 0.001659 | <1% |
| FC-3 | 0.001589 | <1% |
| Total | 127.1163 | 100% |
| 8700K with Parallelism | | |
| Function Segment | Time (Seconds) | Ration in Total Time |
| Data Loading & Pre-process | 9.804942 | 19.74% |
| Conv-1 | 1.803836 | 3.63% |
| Conv-2 | 38.02353 | 76.56% |
| FC-1 | 0.025257 | <1% |
| FC-2 | 0.001316 | <1% |
| FC-3 | 0.000981 | <1% |
| Total | 49.66721 | 100% |

**Table.2a** Running Time on CPU Platform (Non-quantization Version)

|  |  |  |
| --- | --- | --- |
| 8700K without Parallelism | | |
| Function Segment | Time (Seconds) | Ration in Total Time |
| Data Loading & Pre-process | 9.52507 | 6.82% |
| Conv-1 | 3.477169 | 2.49% |
| Conv-2 | 126.61824 | 90.67% |
| FC-1 | 0.02053 | <1% |
| FC-2 | 0.001604 | <1% |
| FC-3 | 0.0013259 | <1% |
| Total | 139.6440 | 100% |

**Table.2b** Running Time on CPU Platform (Quantization Version)

**Fig.9** The Relationship between Batch-Normalization Parameters’ Decimal Bits and Model’s Prediction Accuracy

**5.2 Accelerator’s Implement and Performance**

**Fig. 10** Performance Compared with Multiple CPU Platforms

To implement our layer-by-layer pipeline, it is necessary to expand Conv-1’s degree of parallelism to handle nine 3x3 slide windows on a 5x5 feature area. Not only the Conv-1 layer’s computing scale is expanded, but the data-address generating strategy is also modified to fetch input feature pixel in specific order.

By adapting pipeline on layers, target neural networks can be accelerated without putting between-layers results into DRAM and thus reducing memory accessing cost. In another word, we keep data stream in layer pipeline from input audio feature to final predict results without stop. The accelerator only communicates with DRAM in fetching and final writing back. Inside each layer’s pipeline, we also divide all computing into some function parts like vector computing unit, normalization unit in pipeline method, which can help to rise hardware running frequency.

We run this speech classification model on intel Core i7-8700K (3.7GHz, 6 cores, 95W) with single thread, intel Core i7-8700K with multi-thread and multi-node intel Xeon 5220 (2.2GHz, 18 cores, 125W) with Matlab distributed parallel library, the whole dataset we used contains 1512 audio files. We only test the running time of neural network’s forwarding part for we do not implement the MFCC and data preprocessing on FPGA. Fig.10 shows that compared to state-of-the-art CPU platforms, our single PE version accelerator achieves 18~300x throughput speed up ratio. Table.2 shows that Conv-2 layer is the most time-costing function. However, our accelerator can eliminate bottlenecks on CPU platforms and achieve excellent accelerating performance by using balanced layer-by-layer pipeline. Also, the data stream inside the pipeline reduces the DDR bus communication and eliminates the DRAM limit on CPU platform.

The multi-PE version of our accelerator is implemented on Xilinx Vivado 2018.3 and KU-115 FPGA platforms. We conduct the test of computing accuracy, speed-up ratio and power performance validating hardware design. To ensure reliability, we conduct extensive pressure test for up to two hours. The implement results are shown in Table.3.

We compute the energy efficiency of state-of-the-art CPU platforms and ours. The results in Table.4 show that our accelerator has great energy efficiency improvement on this sound classification task by customizing circuit design and replacing floating data with fixed-point data.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PE | LUT Utilization | FF Utilization | BRAM Utilization | DSP Utilization | Frequency |
| 1 | 25% | 14% | 9% | 12% | 150MHz |
| 2 | 47% | 26% | 10% | 23% | 150MHz |
| 3 | 66% | 38% | 11% | 36% | 150MHz |

**Table.3** Accelerator’s Implementation Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 8700K | 5220-1Node | 5220-2 Node | 5220-4 Node | 1PE BWN | 2PE BWN |
| Perf. Per Watt (fps/W) | 0.23 | 0.47 | 0.43 | 0.4 | 471.63 | 753.41 |
| Board Power (W) | 95W | 125W | 250W | 500W | 7.794W | 9.758W |

**Table.4** Energy Efficiency on Different Platforms

In Table.5, we compare our accelerator with BWCNN-based (Binary Weight Convolutional Neural Network) ASIC architectures, fully binarized CNN FPGA architectures and sparse RNN-based FPGA accelerators.

Table.5a illustrates comparison between BWCNN-based ASIC and ours. Two ASIC architecture own huge advantages on power-consuming and power-efficiency, however, our FPGA platform surpasses these ASIC design in expense, flexibility and perk performance.

The detail information of typical full-binary CNN-based FPGA accelerator is shown in Table.5b, together with ours. Our design performs better in peak performance, model’s accuracy loss and have larger throughput than two of targeted architectures. For fully-binarized accelerators binarizing both weight and activation data, taking less resource (which enables designer to set far more PEs than us), it is considerable for some design outperforming ours in throughput. However, this fully-binarized scheme bears more prediction precision loss, in contrast, ours can keep model’s classification performance better.

Current state-of-the-art RNN-based solutions (Recurrent Neural Network) usually use networks’ sparse feature to cut down useless computation thus accelerating original algorithm. For this reason, sparse accelerators can sometimes have excellent computing performance. Nevertheless, our BWN-based convolutional neural network accelerator still has least computing latency benefitting from CNNs’ simple architecture, while RNNs usually own complex recurrent structures which increase time-delay greatly.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accelerator | Bo L, et al | Renzo A, et al | 1 PE BWN | 3 PE BWN |
| Platform | ASIC@28nm | ASIC@65nm | FPGA | FPGA |
| Frequency | 400MHz | N/A | 150MHz | 150MHz |
| Peak Perf. | 25.6GOPS | 15GOPS@0.6V | 23.85GOPS | 71.55GOPS |

**Table.5a** Comparison with Binary Weight Convolutional Neural Network’s Architecture on ASIC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accelerator | Y. Umuroglu, et al | H. Alemdar, et al | Baicheng L, et al | 3 PE BWN |
| Platform | ZC706 | Kintex-7 160T | ZC706 | XCKU-115 |
| Frequency | 200MHz | 200MHz | 120MHz | 150MHz |
| Peak Perf. | 4.43GOPS | 56.09GOPS | 13.92GOPS | 71.55GOPS |
| Accuracy Loss | 0.64% | 0.75% | 0.41% | 0.35% |
| Throughput | 0.76K fps | 31.8Kfps | 1.4Kfps | 11Kfps |

**Table.5b** Comparison with Full-Binary Convolutional Neural Network’s Architecture on FPGA. \*Peak Perf. And Throughput are equivalent to 16bit fixed.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accelerator | Song H, et al | Shuo W, et al | Shijie C, et al | 3 PE BWN |
| Platform | XCKU060 | Virtex-7 | ZC706 | Arria 10 GX1150 |
| Frequency | 200MHz | 200MHz | 200MHz | 150MHz |
| Perf. at Batch 1 | 8.8GOPS | 43.7GOPS | 304.1GOPS | 71.55GOPS |
| Latency | 15878.4us | 3206.4us | 460.8us | 218.4us |

**Table.5c** Comparison with Sparse Recurrent Neural Network’s Architecture on FPGA. \*Latency is the average time each accelerator needs to process one audio sample.

**6. Conclusion**

We first optimize a sound classification algorithm based on deep convolutional neural network. By quantization method, the activation data size is reduced sharply and time-consuming floating computation is replaced by faster fixed-point computation. Our accelerator design then focuses on parameter-shared storage structure, bitwise expansion and balanced layer-by-layer pipeline. With the combination of deep convolutional neural network quantization and customized circuit design, we bring out a FPGA-based acoustic task accelerator which has high throughput, low time-delay, energy-efficiency and high-performance. Compared with current state-of-the-art CPU platform, BWCNN-based ASIC architectures, fully binarized FPGA accelerators and sparse RNN accelerators, our hardware design has great advantages on computing performance, prediction accuracy, time-delay and power efficiency. We implement our design on Xilinx FPGA, it turns out that this accelerator is a reliable and high-performance intelligent computing platform.

**Acknowledgment**

This work is supported by National Science and Technology Major Project 2018ZX01028101.

**Compliance with ethical standards**

**Conflict of interest**

On behalf of all authors, the corresponding author states that there is no conflict of interest.

**Reference**

Geoffrey H, Li D, Dong Y, et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition [J]. IEEE Signal Processing Magazine, 2012, 12:82-98.

Wan H , Guo S , Yin K , et al. CTS-LSTM: LSTM-based neural networks for correlated time series prediction[J]. Knowledge Based Systems, 2019, 191.

Tom S, Vaibhava G. Advances in Very Deep Convolutionalal Neural Networks for LVCSR[C/OL]. arXiv:1604.01792v2[cs.CL].

Muckenhirn H , Magimai.-Doss M , Marcell S . [IEEE ICASSP 2018 - 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) - Calgary, AB (2018.4.15-2018.4.20)] 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) - Towards Directly Modeling Raw Speech Signal for Speaker Verification Using CNNS[C]// 2018:4884-4888.

Palaz D , Magimai.-Doss M , Collobert R . Convolutionalal neural networks-based continuous speech recognition using raw speech signal[C]// 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2015.

Pakyurek M, Atmis M, Kulac S, et al. Extraction of Novel Features Based on Histograms of MFCCs Used in Emotion Classification from Generated Original Speech Dataset[J]. Electronics & Electrical Engineering, 2020, 26:46-51.

Jyrki Kivinen M K W . Exponentiated Gradient versus Gradient Descent for Linear Predictors[J]. Information and Computation, 1997, 132( 1):1-63.

Perkins S , Lacker K , Theiler J . Grafting: Fast, Incremental Feature Selection by Gradient Descent in Function Space[J]. Journal of Machine Learning Research, 2003, 3(3):1333-1356.

Liu S , Pattabiraman K , Moscibroda T , et al. Flikker: Saving DRAM Refresh-power through Critical Data Partitioning[J]. Computer architecture news, 2011, 39(1):p.213-224.

Han S , Mao H , Dally W J . Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding[C]// ICLR. 2016.

Dundar G , Rose K . The effects of quantization on multilayer neural networks[J]. IEEE Transactions on Neural Networks, 1995, 6(6):1446-1451.

Guo P , Ma H , Chen R , et al. A High-Efficiency FPGA-Based Accelerator for Binarized Neural Networks[J]. Journal of Circuits System & Computers, 2019.

Liang S , Yin S , Liu L , et al. FP-BNN: Binarized neural network on FPGA[J]. Neurocomputing, 2018, 275(JAN.31):1072-1086.

Conti F , Schiavone P D , Benini L . XNOR Neural Engine: A Hardware Accelerator IP for 21.6-fJ/op Binary Neural Network Inference[J]. IEEE Transactions on Computer Aided Design of Integrated Circuits & Systems, 2018, 37(11):2940-2951.

Michael D, Ashish K, David R. Nvidia’s Xavier Soc[C]. IEEE Hot Chips 2018.

Nvidia. Deep Learning Accelerator[C]. IEEE Hot Chips 2018.

Gwennap L . Microsoft Brainwave Uses FPGAs[J]. Microprocessor Report, 2017, 31(11):25-27.

Turan F , Roy S S , Verbauwhede I . HEAWS: An Accelerator for Homomorphic Encryption on the Amazon AWS FPGA[J]. IEEE Transactions on Computers, 2020, PP(99):1-1.

Bo L, Hai Q, Yu G, et al. EERA-ASR: An Energy-Efficient Reconfigurable Architecture for Automatic Speech Recognition with Hybrid DNN and Approximate Computing[J]. IEEE ACCESS, 2018, 6:52227-52237.

Matthieu C, Itay H, Daniel S, et al. Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1[C/OL]. arXiv:1602.02830v3[cs.LG].

Jacob B , Kligys S , Chen B , et al. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference[J]. 2017.

Xu Y , Wang Y , Zhou A , et al. Deep Neural Network Compression with Single and Multiple Level Quantization[J]. 2018.

Santurkar S , Tsipras D , Ilyas A , et al. How Does Batch Normalization Help Optimization [J]. 2018.

Liu M , Wu W , Gu Z , et al. Deep Learning Based on Batch Normalization for P300 Signal Detection[J]. Neurocomputing, 2017:S0925231217314601.

Giri E P , Fanany M I , Arymurthy A M , et al. Ischemic Stroke Identification Based on EEG and EOG using 1D Convolutionalal Neural Network and Batch Normalization[C]// ICACSIS. IEEE, 2016.

Cheung K , Schultz S R , Luk W . A Large-Scale Spiking Neural Network Accelerator for FPGA Systems[C]// Proceedings of the 22nd international conference on Artificial Neural Networks and Machine Learning - Volume Part I. Springer, Berlin, Heidelberg, 2012.

Alessandro A , Hesham M , Enrico C , et al. NullHop: A Flexible Convolutionalal Neural Network Accelerator Based on Sparse Representations of Feature Maps[J]. IEEE Transactions on Neural Networks & Learning Systems, 2018:1-13.

Chen T , Du Z , Sun N , et al. A High-Throughput Neural Network Accelerator[J]. IEEE Micro, 2015, 35(3):24-32.

Wei Z , Jingyi Q , Renbiao W . Straight Convolutionalal Neural Networks Algorithm Based on Batch Normalization for Image Classification[J]. Journal of Computer-Aided Design & Computer Graphics, 2017.

Simonyan K , Zisserman A . Very Deep Convolutionalal Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.

Krizhevsky A , Sutskever I , Hinton G . ImageNet Classification with Deep Convolutionalal Neural Networks[C]// NIPS. Curran Associates Inc. 2012.

Zeng X, Zhi T, Zhou X, Du Z, Guo Q, Liu S, et al. Addressing Irregularity in Sparse Neural Networks Through a Cooperative Software/Hardware Approach. IEEE Transactions on Computers[J], 69(7):968–85, 2020.

Cheng G, Yao C Ye L, Tao L, Cong H, et al. Vecq: Minimal loss DNN model compression with vectorized weight quantization. IEEE Transactions on Computers[J], 2020.

Renzo A, Lukas C, Davide R, Luca B, YodaNN: An Architecture for Ultralow Power Binary-Weight CNN Acceleration. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems[J], Vol:37, Issue: 1, 2018.

Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre,and K. Vissers, Finn: A framework for fast, scalable binarized neuralnetwork inference, ACM/SIGDA Interna-tional Symposium on Field-Programmable Gate Arrays[C], pp. 65–74, 2017.

H. Alemdar, V. Leroy, A. Prost-Boucle, and F. P ́etrot, Ternary neuralnetworks for resource-efficient ai applications, International Joint Conference on Neural Networks[C] (IJCNN), pp. 2547–2554, 2017.

Baicheng L, Song C, Yi K, Feng W, et al. An Energy-Efficient Systolic Pipeline Architecture for binary Convolutionalal Neural Network. IEEE International Conference on ASIC[C], 2019.

Song H, Junlong K, Huizi M, et al. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA. FPGA’17: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays[C], pp.75-84, 2017

Shuo W, Zhe L, Caiwen D, et al. C-LSTM: Enabling Efficient LSTM using Structured Compression Techniques on FPGAs. FPGA’18: Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays[C], pp.21-30, 2018

Shijie C, Chen Z, Zhuliang Y, et al. Efficient and Effective Sparse LSTM on FPGA with Bank-Balanced Sparsity. FPGA’19: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays[C], pp.63-72, 2019

1. National University of Defense Technology, Changsha, China

   \* Corresponding Author. Corresponding E-mail Address: jingfeijiang@nudt.edu.cn [↑](#footnote-ref-1)