Highly Efficient SNNs for High-speed Object Detection

Nemin Qiu[1]
qiunm@bupt.edu.cn
Zhiguo Li[2]
youxianglizhiguo@163.com
Yuan Li[3]
yuanli@pku.edu.cn
Chuang Zhu*
czhu@bupt.edu.cn

[1] School of Artificial Intelligence, Beijing University of Posts and Telecommunications Beijing, China
[2] Peking University Beijing, China

Abstract

The high biological properties and low energy consumption of Spiking Neural Networks (SNNs) have brought much attention in recent years. However, the converted SNNs generally need large time steps to achieve satisfactory performance, which will result in high inference latency and computational resources increase. In this work, we propose a highly efficient and fast SNN for object detection. First, we build an initial compact ANN by using quantization training method of convolution layer fold batch normalization layer and neural network modification. Second, we theoretically analyze how to obtain the low complexity SNN correctly. Then, we propose a scale-aware pseudo-quantization scheme to guarantee the correctness of the compact ANN to SNN. Third, we propose a continuous inference scheme by using a Feed-Forward Integrate-and-Fire (FewdIF) neuron to realize high-speed object detection. Experimental results show that our efficient SNN can achieve 118× speedup on GPU with only 1.5MB parameters for object detection tasks. We further verify our SNN on FPGA platform and the proposed model can achieve 800+FPS object detection with extremely low latency.

1 Introduction

Artificial Neural Networks (ANNs) have achieved great success in computer vision [19], natural language processing[25] and other fields. However, the success of ANNs is also accompanied by some serious concerns on their huge demand on computational resources and power consumption. In contrast, the human brain can provide excellent cognitive abilities with ultra-low natural power. Thus, many brain-inspired Spiking Neural Networks (SNNs)[4, 28] are proposed to decrease computational resources and power consumption. SNNs are viewed as the third generation of neural network models, using biologically-realistic but simplified models of neurons to carry out computation. The event-driven mechanism in SNNs greatly avoids consuming excessive resources to a large extent[8]. SNNs are suitable to be implemented on low-power mobile or edge devices [9, 13].

Corner symbol '*' in the author name means the corresponding author.

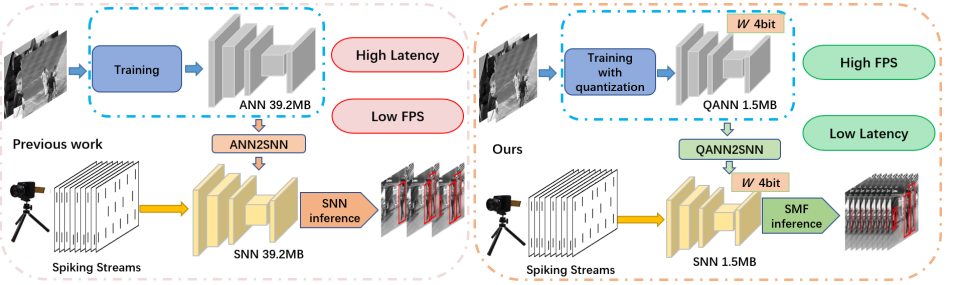


Figure 1: Illustration of our method and the previous work for SNN object detection. Previous work get the SNN model (39.2MB) by converting trained ANN with 32-bit precision[8, 21] (ANN2SNN). It causes high latency and low FPS. Our method first obtains an initial quantized ANN (QANN), and then obtains a model size of only 1.5MB SNN with 4-bit precision by QANN2SNN conversion. FPS can be further increased using our proposed SNN continuous inference. We deploy and implement 800+FPS detection on FPGA using only 2.4W of power.

At present, direct training SNNs and ANN to SNN conversion are two ways to generate SNN model. The SNN model obtained by direct training suffers unsatisfactory accuracy[26] due to the use of surrogate gradient[23, 24, 28] to address the non-differentiable binary activation function. The converted SNNs can obtain satisfactory performance, and we focus on this kind of SNN model in this paper. However, to maintain decent model precision, the converted SNNs generally need large time steps (such as work[8] taking thousands of time steps in object detection), which result in high inference latency[12, 20, 26] and computational resources increase[20, 27]. Moreover, the converted SNNs still suffer large model size due to the corresponding high complex ANNs. Fig. 1 illustrates the large SNN model of the previous works[8, 21] need to accumulate many time steps to achieve decent performance, which result in low FPS (Frames Per Second).

In this work, we propose a highly efficient and fast SNN for object detection. First, we build an initial compact ANN by using quantization training method of convolution layer fold batch normalization layer and neural network modification. Second, we theoretically analyze how to obtain the low complexity SNN correctly by using conversion method. Meanwhile, we propose a scale-aware pseudo-quantization scheme to guarantee the correctness of the quantized ANN to SNN. Then we obtain a highly efficient low complexity SNN. Third, we propose a continuous inference scheme to realize high-speed object detection. Specifically, to support our continuous inference, we design a Feed-Forward Integrate-and-Fire (FewdIF) neuron which is capable of accumulating history information.

To summarize, our main contributions are as follows:

- We propose a highly efficient and fast SNN for object detection. Specifically, we first convert the quantized ANN to low complexity SNN, and then construct a continuous inference scheme to realize high-speed object detection.

- In the SNN conversion, we first perform quantization training method of convolution layer fold batch normalization layer and neural network modification. Then, we propose a scale-aware pseudo-quantization scheme to guarantee the correctness of the quantized ANN to SNN.

- In the inference stage, we propose a continuous inference scheme to realize high-speed object detection by using our designed FewdIF neuron.

- Experimental results show that our efficient SNNs have few and low bit-width parameters (1.5MB) and high-speed detection (GPU: FPS 177.5 vs 1.5[8]) on object detection tasks. We further deploy the SNNs on FPGA and achieve 800+FPS detection with extremely low latency.

2 Related Work

ANN to SNN Conversion: The conversion of ANN to SNN is in burgeoning research. Cao *et al*. [2] proposed a ANN to SNN conversion method that neglected bias and max-pooling. In the next work, Rueckauer *et al*.[21] presented an implementation method of batch normalization and spike max-pooling. Meanwhile, To get deeper SNNs. Diehl *et al*.[3] proposed the data-based normalization to improve the performance in deep SNNs. Sengupta *et al*.[22] expanded conversion methods to VGG and residual architectures. However, the converted SNN requires massive time steps to reach competitive performance[20]. All of them are complicated procedures vulnerable to high inference latency[17, 26]. To reduce the time step, Park *et al*.[17] proposed a fast and energy-efficient information transmission method with burst spikes and hybrid neural coding scheme in deep SNNs. Ding *et al*.[4] presented Rate Norm Layer to replace the ReLU function, and obtain the scale through a gradient-based algorithm. Nonetheless, most previous works have been limited to the image classification task.

Object Detection for SNN: Kim *et al*.[8] have presented Spiking-YOLO, the first SNN model that successfully performs object detection by achieving comparable results to those of the original ANNs on non-trivial datasets, PASCAL VOC and MS COCO. However, it suffers from high inference latency and computational resources increase[26]. Moreover, the converted SNNs still suffer large model size due to the corresponding high complex ANNs.

Model Compression: In the field of pruning neural networks, the pruning methods [14, 29] usually compresses the model and accelerates the inference. In the field of of quantization. Jacob *et al*.[6] propose a quantization scheme that relies only on integer arithmetic to approximate the floating-point computations in a neural network.

There are no efforts to compress and accelerate SNNs on detection tasks. In this paper, based on the existing work, we further design a highly efficient and fast SNN for object detection.

3 Method

In this section, we will present how we implement the highly efficient and fast SNN from two stages, generation and inference, respectively. Fig. 2 is the overview of the generation stage and inference stage. The generation stage contains the quantization training of the initial compact ANN (QANN) in Section 3.1 and the conversion of the quantized ANN to the low complexity SNN (QANN2SNN) in Section 3.2. The inference stage includes Feed-Forward Integrate-and-Fire (FewdIF) neurons and SNN continuous inference we proposed in Section 3.3.

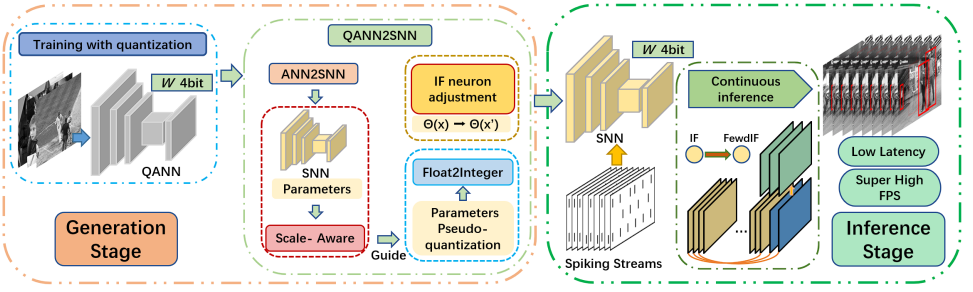


Figure 2: The Illustration for achieving super high FPS and low latency object detection by using the proposed highly efficient SNN. There are two stages. In generation stage, we build an initial compact quantized ANN (QANN) by using quantization training method of convolution layer fold batch normalization layer and neural network modification. Then we correctly convert the quantized ANN to SNN (QANN2SNN) by using our proposed scale-aware pseudo-quantization scheme. Thus we get a highly efficient low complexity SNN. In inference stage, replacing the original IF neurons with proposed FewdIF neurons can achieve the proposed SNN continuous inference which can greatly improve FPS.

3.1 ANN Quantization

In this section, we focus on the preparing work for efficient SNN generation. We build an initial compact ANN by using quantization training method of convolution layer fold batch normalization layer and neural network modification.

Specifically, we reduce the bit-width of the ANN weights by using quantization. The low bit-width compact ANNs can be correctly converted to SNN by using QANN2SNN method in Section 3.2. This allows the weight bit-width of the converted SNN to be further reduced as well. Considering that the performance of the converted SNN depends on its initial ANN [16], we need to build an initial compact ANN that performs well and is suitable for conversion to SNN.

We build the initial compact quantized ANN by using the training in the Fig. 3. It is a training method adapted for QANN2SNN method. In particular, we generate initial compact quantized ANN by using quantization training method of convolution layer fold batch normalization layer and neural network modification. Noteworthy, to obtain better ANN, we use training with simulated quantization [6, 10] as our method of quantization training. What's more, we train with Quantized ReLU (QReLU) instead of ReLU, which not only completes the operation of quantizing activation values, but also reduces the time steps of SNN [4]. For better SNN performance after conversion, we use down-sampling convolution to replace max-pooling layer. The upsampling layer in ANN is replaced by transpose convolution.

3.2 Quantized ANN to SNN

In order to avoid the bit-width rise of SNN weights after the conversion, we propose a scale-aware pseudo-quantization scheme to guarantee the correctness of the quantized ANN (QANN) to SNN. The conversion of QANN to SNN consists of the following three steps: weight conversion, weight bit-width mapping and type conversion, and IF neuron adjust-

ment. To simplify the description, in this section, let $Q(\cdot)$ denotes the int8 quantization function.

ANN to SNN: The similarity of Integrate-and-Fire (IF) neuron and ReLU activation functions[21] is an important basis on which ANNs can be converted to SNNs. The principle of ANN to SNN conversion is that the firing rates of spiking neuron $r_k^l(T)$ should correlate with the original ANN activations x_k^l such that $r_k^l(T) \rightarrow x_k^l$. The firing rate of each SNN neuron as $r_k^l(T) = N_k^l(T)/T$, where $N_k^l(T) = \sum_{t=1}^{T} \Theta_{t,k}^l$ is the number of spikes generated in T time steps, let's $\Theta_{t,k}^l$ denotes a step function indicating the occurrence of a spike at time t. For activation function mapping $r_k^l(T) \rightarrow x_k^l$ in ANN to SNN conversion, there has been a lot of previous works[4, 8, 21] that describes this (Theory for Conversion from ANN to SNN) in detail. The layer-to-layer relationship between the firing rates of IF neurons obtained through a series of derivations and approximations is:

$$r_k^l(T) \approx \sum_j (w_{k,j}^l \cdot r_j^{l-1}(T)) + b_k^l. \tag{1}$$

This relationship is very similar to the ANN's layer-to-layer activation value relationship:

$$x_k^l = \sum_j (w_{k,j}^l \cdot x_j^{l-1}) + b_k^l. \tag{2}$$

Weight conversion: From the above definition, it is clear that the firing rate of IF neurons $r_k^l(T) \in [0,1]$, we need to adjust the output range of ANNs ReLU activation function to [0, 1][21]. Therefore, we achieve this adjustment by converting the parameters of the ANNs. The well-known layer-wise parameter normalization (LayerNorm)[21] is a typical parameters transformation method. Specifically, after the ANN model is trained, we need to count the input tensor and output tensor of this layer. The maximum value of the input tensor is M^{l-1}, the maximum value of the output tensor is M^l, and the normalized weight and bias should be as follow:

$$\hat{w}_{k,j}^l = \frac{w_{k,j}^l \cdot M^{l-1}}{M^l}, \qquad \hat{b}_k^l = \frac{b_k^l}{M^l}. \tag{3}$$

where $w_{k,j}^l$ represents weights, b_k^l represents biases. After completing the above operations, replace the ReLU activation function in the ANN with the IF neuron.

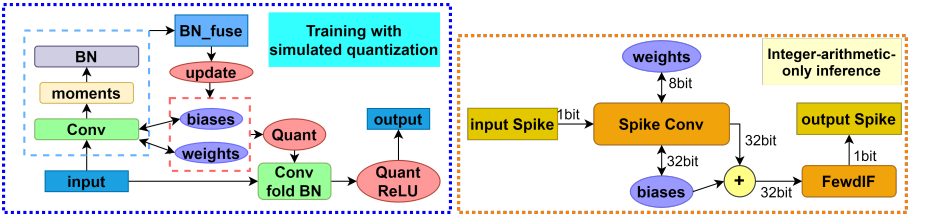
In order to get the SNNs with low bit-width parameters, let us introduce the equation for quantized ANNs x_k^l as shown in Eq. (4).

$$Q(x_k^l) = f(\sum_{j=0}^{n} (Q(w_{k,j}^l) \cdot x_j^{l-1}) + Q(b_k^l)). \tag{4}$$

By using the previous ANN to SNN conversion method, some adjustments are made to the Eq. (3). The maximum value of the input tensor after quantization is $Q(M^{l-1})$, the maximum value of the output tensor after quantization is $Q(M^l)$, and the normalized weights and biases should be as follow:

$$\hat{w}_{k,j}^l = \frac{Q(w_{k,j}^l) \cdot Q(M^{l-1})}{Q(M^l)}, \qquad \hat{b}_k^l = \frac{Q(b_k^l)}{Q(M^l)}. \tag{5}$$

We find a problem encountered in converting the quantized ANN to SNN according to Eq. (5). Specifically, the converted weights $\hat{w}_{k,j}^l$ and biases $\hat{b}_k^l$ are obtained by multiplying corresponding int8 values according to Eq. (5). The bit-width of parameters is obviously



(a) Train (b) Infer

Figure 3: Quantization training of initial compact ANN and Integer-arithmetic-only inference of the low complexity SNN.

increased, because two operations with high bit numbers require higher bits to store lossless results.

Weight bit-width mapping and type conversion: To solve the above problems, we propose a scale-aware pseudo-quantization scheme to guarantee the correctness of the quantized ANN to SNN. We divide them by the minimum interval of two numbers, and these parameters can still be stored with int8 bit-width. Let $U_k^l(t)$ denote a transient membrane potential increment of spiking neuron k in layer l, Our method uses $\hat{U}_k^l(t)$ instead of $U_k^l(t)$:

$$\hat{U}_k^l(t) = \sum_j (\hat{w}_{k,j}^l \cdot S_l \cdot \Theta_{t,j}^{l-1}) + \hat{b}_k^l \cdot S_l = U_k^l(t) \cdot S_l, \tag{6}$$

lets s_l denotes the minimum of the absolute value of the difference between any of the weight values in l layer. Where S_l represents $S_l = 1/s_l$. $\Theta_{t,k}^l$ denotes the output of the spiking neuron k at moment t. By type conversion we can get the integer $Int(w_{k,j})$:

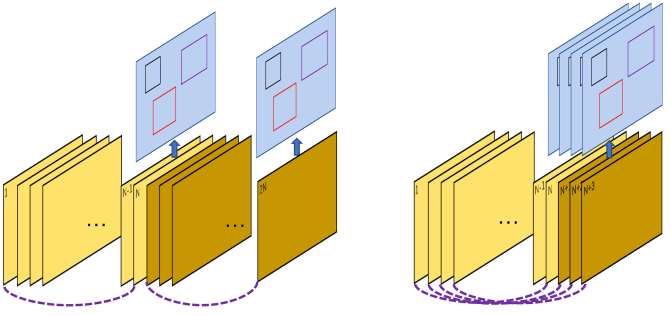
$$Int(w_{k,j}) = Int(\hat{w}_{k,j}^l \cdot S_l). \tag{7}$$

$Int(w_{k,j})$ will be used in the inference. For the biases, according to Eq. (5), it is known that the minimum interval of the bias is not necessarily the same as the minimum interval of the weights. Therefore, for the biases, we use 32-bit floating point storage or direct rounding to 32-bit int type. Although the biases are quantized as 32-bit values, they account for only a tiny fraction of the parameters in a neural network [□].

IF neuron adjustment: According to the definition of spiking neuron output $\Theta_{t,j}^l$, the spiking neuron integrates inputs $U_k^l(t)$ until the membrane potential $V_k^l(t-1)$ exceeds a threshold $V_{k,th}$ and a spike is generated. In the case of using our method, if we want to ensure that the output of $\Theta_{t,j}^l$ is not affected by linear change $\hat{U}_k^l(t)$, we need to multiply $V_{k,th}$ also with the scale factor S_l:

$$\Theta_{t,k}^l = \Theta(V_k^l(t-1) + \frac{(\hat{U}_k^l(t) - S_l \cdot V_{k,th})}{S_l}). \tag{8}$$

After finishing these corrections, we successfully solve the problem of converting a low bit-width ANN to a low bit-width SNN. Finally, we can use the SNN Integer-arithmetic-only inference architecture shown in Fig. 3. Experimental results show that our efficient SNNs with few and low bit-width parameters overcome high latency on object detection tasks. Compared to previous methods, our SNN model with 4-bit parameters exceeds the performance of previous methods using few time steps.



(a) SNN Inference (b) SNN Continuous Inference

Figure 4: Comparison of (a) SNN inference and (b) Continuous inference. With the same number of spiking datas, more results are output using continuous inference.

3.3 SNN Continuous Inference

Most of the previous ANN to SNN works focus on single image tasks. Their SNN inference [8, 21] is shown in Figure 4 (a). SNN inferring one frame result need to accumulate N frames of spike data each time to correspond to the output one frame result as ANN. The neuronal membrane potential of IF neuron is reset to 0 after every N time steps. Considering the continuous scenario, we believe that such an inference approach does not make good use of the spiking data. We propose a continuous inference scheme shown in Figure 4 (b). We do not set the membrane potential to 0 in the continuous scenario. In this way, the first frame of the SNN output needs the input of the spike frames from the first frame to N_{th} frame. While the second only needs the input of new $(N+1)_{th}$ spike frame to predict a result. It is different from the previous inference which needs the input of $(N+1)_{th}$ frame to $2N_{th}$ frame. However, IF neurons use this method with severe performance degradation.

To solve the above problems, we propose Feed-Forward Integrate-and-Fire (FewdIF) neurons to avoid excessive "excitation" and "inhibition" of IF neurons. The purpose of the modifications is to limit the maximum and minimum accumulation of membrane potentials to ensure that the previous frame may affect the input of the next frame, but not "overcall". The positive and negative boundary values of the two membrane potentials that a neuron's membrane potential can accumulate to at most is defined as follows:

$$MAX(V_k^l(t))_{FewdIF} = (N_{max} \cdot V_{k,th}), \tag{9}$$

$$MIN(V_k^l(t))_{FewdIF} = (N_{min} \cdot V_{k,th}), \tag{10}$$

where N_{max} is the maximum scale factor and N_{min} is the minimum scale factor of FewdIF neuron. By using FewdIF neurons instead of IF neurons in an SNN, continuous inference can be achieved with a single time step after the SNN has adapted to the scenario.

Experiments show that the SNN continuous inference only need one time step to predict. Compared to the previous work on object detection [8], we significantly reduce the time steps.

4 Experiments

Since there are almost no researches in this area yet, we did our best to conduct the comparison with relevant experiments[1, 8, 12, 15]. In the Section 4.1, we set up a performance comparison of two SNNs on object detection task to validate our low complexity SNN obtained in Section 3.2. In the Section 4.2, we set up a comprehensive comparison of different SNN inference methods to verify our FewdIF neuron and SNN continuous inference proposed in Section 3.3. Section 4.3 shows the ultra-high power efficiency of our SNN on FPGA.

We select some videos from the MOT challenge [11] as the validation dataset for our experiments. The spike datasets involved in this work are spiking streams in continuous scenes, which are captured using spiking cameras or by encoding the video with spike encoder [5, 8].

The detection results of our experiments are evaluated using mAP50(%). The experiments are performed on Ubuntu system. Our simulation is based on the Pytorch framework and we conducted all experiments on NVIDIA Tesla V100 32G GPUs.

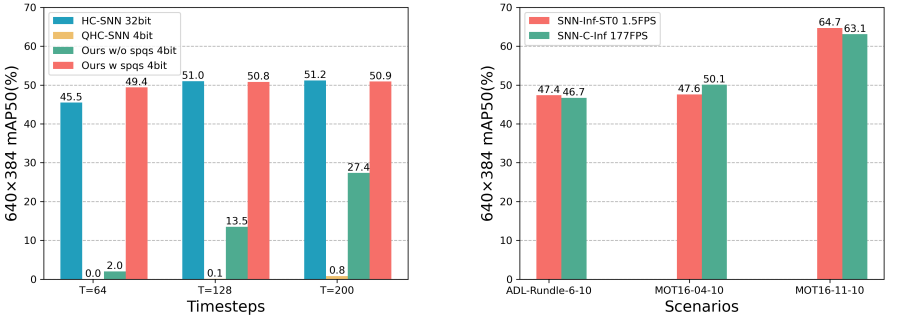
4.1 Comparison of the Two SNNs

Since the previous method [8] is not open source, we use an improved version (high complexity SNN (HC-SNN[8])) to represent it. The first experiment explores the performance between high complexity SNN (39.2MB) and low complexity SNN (1.5MB). High complexity SNN (HC-SNN[8]) uses 32-bit floating point precision to store the weights, which is converted from high complexity ANN (HC-ANN[8]) by using ANN to SNN conversion methods [4, 8, 12, 25]. HC-ANN[8] is trained according to the previous methods [8] [12] etc. Its network architecture is similar to that of tiny-yolov3. Our low complexity SNN (**Ours**) uses 4-bit integer precision to store the weights, which is converted from the initial compact quantized ANN (**QANN**) by using our method in Section 3.1. We compare the size of the input data at 640×384. Similarly we also compare in the cases of time steps T=64, T=128, T=200 and T=256 respectively. The Table 1 shows the results when the input data size is 640×384. In addition we tried to prune and directly quantize the HC-SNN[8] (QHC-SNN) to reduce the bit-width of the weights, but there is a complete loss of performance. We also performed ablation experiments, whether to use our proposed scale-aware pseudo-quantization scheme(spqs) or not. The accuracy after compression by different methods is shown in Fig. 5. We can summarize the following conclusions:

First, the results of HC-SNN[8] show that our HC-SNN[8] have good performance. Compared to the previous method [8] which takes thousands of time steps, we need only 64 time steps for SNN object detection, even exceeding the performance of the original ANN in some scenarios. Second, compared to the direct quantification scheme, the performance of the low complexity SNN obtained by our method is much better than the QHC-SNN which has the same model size. And after using our proposed scale-aware pseudo-quantization scheme, the performance is almost lossless compared to the HC-SNN[8]. The results show that the performance of our low complexity SNN is comparable to the initial compact quantized ANN, and even better than QANN in some scenarios. By carefully comparing the mAP50 in Table 1, we can find that our low complexity SNN can achieve very close performance to HC-SNN[8] in all time step cases and all scenarios. However, our model size is only 1/26 of the original model size. This is an important reason why we can deploy it to FPGA.

640×384 mAP50(%)	ANN		T=64		T=128		T=200		T=256	
Model size	39.2MB	**1.5MB**	39.2MB	**1.5MB**	39.2MB	**1.5MB**	39.2MB	**1.5MB**	39.2MB	**1.5MB**
Scenarios	HC-ANN	QANN	HC-SNN	**Ours**	HC-SNN	**Ours**	HC-SNN	**Ours**	HC-SNN	**Ours**
ADL-Rundle-6	50.2	49.8	45.5	49.4	51.0	50.8	51.2	50.9	51.4	50.9
ADL-Rundle-6_gray	50.2	49.8	42.5	49.4	51.2	50.7	51.4	50.5	51.3	50.7
MOT-11	60.8	62.2	27.0	59.7	59.0	61.5	60.3	61.9	60.5	62.0
AVG_TownCentre	53.1	50.4	44.4	47.8	50.8	50.2	53.7	50.2	53.9	50.4
ADL-Rundle-8	43.3	40.9	37.9	37.0	44.0	38.4	44.0	38.4	44.4	38.5

Table 1: Comparison of the object detection performance(640×384 pixels mAP50) between our low complexity SNN (Ours) and the high complexity SNN (HC-SNN[8]) in different scenarios.



(a) Accuracy after compression by different methods (b) Performance of the two inference methods

Figure 5: (a): Comparison of direct quantize the HC-SNN[8] (QHC-SNN) and our methods (w or w/o spqs). (b): Performance comparison of SNN-Inf-ST0 and SNN-C-Inf.

4.2 Comparison of Different Inference Methods

In this experiment, the input spiking data size is 640×384 and the default time steps is T=200. Part of the experiment was tested on both SNN models (**Ours** and **HC-SNN[8]**). Our experiment is set up with two types of inference. The first inference is the previous method using SNN inference (SNN-Inf-ST0[7]) with IF neurons, the second is the proposed SNN continuous inference (**SNN-C-Inf**).

Fig. 5 shows the performance comparison of SNN-Inf-ST0[7] method and SNN-C-Inf method on our model. The experiment is to test the time consumed by using SNN inference and SNN inference continuous inference on GPU (32-bit floating-point inference), comparing the two methods running on the computer while outputting the same number of frames results. After conducting multiple tests in various scenarios, the average FPS for the two inference methods are as follows: SNN-C-INF: 177 and SNN-Inf-ST0[7]: 1.5. Our approach has significant advantages, and at the same time our accuracy of SNN-C-INF inference can be almost equal to that of the SNN-Inf-ST0[7].

4.3 Comparison of the Power Efficiency

To verify the performance of our highly efficient and fast SNN in the application, we deploy it into FPGA. Most weights of the deployed SNN network are stored using 4-bit. To the best of our knowledge, we are the first experiment to deploy a SNN for object detection to FPGA. Thus, we compared it to previous work on implementing ANN on GPU[18] or FPGA[1, 15].

Method	Sim yolo v2[13]	Tiny yolo v2[15]	Tiny yolo v3[1]	LeNet-SNN*[7]	Ours
Platform	GPU	FPGA	FPGA	FPGA	FPGA
Frequency	1 GHz	200 MHz	200 MHz	150 MHz	150 MHz
GOP	6.5	2.64	2.1	0.22	1.4
Weight bit	32	1	1	8	4
Activation bit	32	6	18	1	1
Image Size	256×256	256×256	256×256	10×28×28	256×256
FPS	232	176	219	164	**681**
Throughput(GOPS)	1512	464.7	460.8	36.08	954
Power(W)	170	8.7	4.81	4.6	**2.437**
Power efficiency(GOP/s/W)	8.89	53.29	95.8	7.84	**391.46**
Power efficiency(FPS/W)	1.365	20.23	45.53	35.65	**279.44**

Table 2: Comparison of the power efficiency of GPU or FPGA deployments.

We also compared it with LeNet-SNN[7] for classification tasks on FPGA. We set the input image resolutions of 256×256. For method[7], their inference takes 10 time steps and the input is a 28×28 MINST picture. We did not evaluate and compare performance due to the different tasks of the comparison methods[1, 2, 15, 18]. We compared the throughput (GOPS), power (W), and power efficiency (GOPS/W or FPS/W) of the devices. Table 4.3 shows the resources used and the power consumption achieved by each method. Experimental results show that we only need 2.437W of power consumption to achieve a detection speed of 681 FPS when the input image size is 256×256. There is a huge improvement in power efficiency compared to the previous methods[1, 7, 15, 18]. If we set the input size as 224×224, experimental results show that the detection speed will even be increased to more than 800FPS.

5 Conclusion

This paper is dedicated to the research of extremely efficient SNN to achieve super high-speed inference. Specifically, we first generate an initial compact quantized ANN and convert it to a low complexity SNN, and then construct a SNN continuous inference scheme to realize high-speed object detection. Since there are almost no researches in these areas yet, we did our best to conduct the comparison with relevant experiments[1, 2, 8, 15]. Our generation method compresses the model size 26× times and helps to restore model accuracy to near-identical levels as the original[8] at the same time. In addition, our inference scheme helps to improve the information utilization and inference speed of SNN. For the application, we implement the first SNN for object detection on the FPGA platform. Beyond the object detection task, the proposed methods are theoretically generalizable to other SNN tasks.

6 Acknowledgements

This work was supported by the National Key RD Program of China (2021ZD0109800). This work was supported by the National Natural Science Foundation of China (81972248).

References

[1] Afzal Ahmad, Muhammad Adeel Pasha, and Ghulam Jilani Raza. Accelerating tiny yolov3 using fpga-based hardware/software co-design. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020.

[2] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015.

[3] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International joint conference on neural networks (IJCNN)*, pages 1–8. ieee, 2015.

[4] Jianhao Ding, Zhaofei Yu, Yonghong Tian, and Tiejun Huang. Optimal ann-snn conversion for fast and accurate inference in deep spiking neural networks. *arXiv preprint arXiv:2105.11654*, 2021.

[5] Liwen Hu, Rui Zhao, Ziluo Ding, Lei Ma, Boxin Shi, Ruiqin Xiong, and Tiejun Huang. Optical flow estimation for spiking camera. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17844–17853, 2022.

[6] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[7] Xiping Ju, Biao Fang, Rui Yan, Xiaoliang Xu, and Huajin Tang. An fpga implementation of deep spiking neural networks for low-power and fast classification. *Neural computation*, 32(1):182–204, 2020.

[8] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: spiking neural network for energy-efficient object detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 11270–11277, 2020.

[9] Youngeun Kim, Yuhang Li, Hyoungseob Park, Yeshwanth Venkatesha, Ruokai Yin, and Priyadarshini Panda. Exploring lottery ticket hypothesis in spiking neural networks. In *European Conference on Computer Vision*, pages 102–120. Springer, 2022.

[10] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[11] Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler. Motchallenge 2015: Towards a benchmark for multi-target tracking. *arXiv preprint arXiv:1504.01942*, 2015.

[12] Yang Li, Xiang He, Yiting Dong, Qingqun Kong, and Yi Zeng. Spike calibration: Fast and accurate conversion of spiking neural network for object detection and segmentation. *arXiv preprint arXiv:2207.02702*, 2022.

[13] Fangxin Liu, Wenbo Zhao, Yongbiao Chen, Zongwu Wang, and Fei Dai. Dynsnn: A dynamic approach to reduce redundancy in spiking neural networks. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2130–2134. IEEE, 2022.

[14] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

[15] Duy Thanh Nguyen, Tuan Nghia Nguyen, Hyun Kim, and Hyuk-Jae Lee. A high-throughput and power-efficient fpga implementation of yolo cnn for object detection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(8):1861–1873, 2019.

[16] Sathish Panchapakesan, Zhenman Fang, and Jian Li. Syncnn: Evaluating and accelerating spiking neural networks on fpgas. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2022.

[17] Seongsik Park, Seijoon Kim, Hyeokjun Choe, and Sungroh Yoon. Fast and efficient information transmission with burst spikes in deep spiking neural networks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[18] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[19] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[20] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.

[21] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017.

[22] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

[23] Sumit B Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. *Advances in neural information processing systems*, 31, 2018.

[24] Amirhossein Tavanaei and Anthony Maida. Bp-stdp: Approximating backpropagation using spike timing dependent plasticity. *Neurocomputing*, 330:39–47, 2019.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[26] Yuchen Wang, Malu Zhang, Yi Chen, and Hong Qu. Signed neuron with memory: Towards simple, accurate and high-efficient ann-snn conversion. In *International Joint Conference on Artificial Intelligence*, 2022.

[27] Zhehui Wang, Xiaozhe Gu, Rick Siow Mong Goh, Joey Tianyi Zhou, and Tao Luo. Efficient spiking neural networks with radix encoding. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[28] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and Luping Shi. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1311–1318, 2019.

[29] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.