electronics

MDPI

Article

Spiking VGG7: Deep Convolutional Spiking Neural Network with Direct Training for Object Recognition

Shuiying Xiang [1,*], Shuqing Jiang [1], Xiaosong Liu [1], Tao Zhang [1] and Licun Yu [2,3]

[1] State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China; sqjiang@stu.xidian.edu.cn (S.J.); xsl2011@aliyun.com (X.L.); taozhang116@163.com (T.Z.)
[2] School of Highway, Chang'an University, Xi'an 710064, China; yulicun1026@163.com
[3] CCCC First Highway Consultants Co., Ltd., Xi'an 710075, China
* Correspondence: syxiang@xidian.edu.cn

Abstract: We propose a deep convolutional spiking neural network (DCSNN) with direct training to classify concrete bridge damage in a real engineering environment. The leaky-integrate-and-fire (LIF) neuron model is employed in our DCSNN that is similar to VGG. Poisson encoding and convolution encoding strategies are considered. The gradient surrogate method is introduced to realize the supervised training for the DCSNN. In addition, we have examined the effect of observation time step on the network performance. The testing performance for two different spike encoding strategies are compared. The results show that the DCSNN using gradient surrogate method can achieve a performance of 97.83%, which is comparable to traditional CNN. We also present a comparison with STDP-based unsupervised learning and a converted algorithm, and the proposed DCSNN is proved to have the best performance. To demonstrate the generalization performance of the model, we also use a public dataset for comparison. This work paves the way for the practical engineering applications of the deep SNNs.

Keywords: deep convolutional spiking neural networks; surrogate gradient; bridge damage detection

Citation: Xiang, S.; Jiang, S.; Liu, X.; Zhang, T.; Yu, L. Spiking VGG7: Deep Convolutional Spiking Neural Network with Direct Training for Object Recognition. *Electronics* **2022**, *11*, 2097. https://doi.org/10.3390/electronics11132097

Academic Editor: Young Min Song

Received: 11 May 2022
Accepted: 1 July 2022
Published: 4 July 2022

1. Introduction

Neural networks (NN) and deep learning (DL) techniques have attracted lots of attention in various application fields including image recognition, detection, and speech recognition. Brain-inspired spiking neural network (SNN), which is called the third generation of NN, is more biologically plausible than the first and second generation of NN [1]. The SNN has the advantages of low-latency and of being energy-efficient by the event-based computation; however, training an SNN is a difficult task due to the non-differentiable nature of these spike events [2,3].

In recent years, tremendous efforts have been devoted to the training algorithms of SNN. Spike-timing-dependent plasticity (STDP) is a popular rule mainly for unsupervised learning [4–9]. The STDP rule has been theoretically and experimentally demonstrated in optical elements [7,8]. A remarkable work of STDP-based unsupervised spiking deep convolutional neural network for object recognition is proposed in [9]. There are also various well-known supervised training algorithms such as SpikeProp [10], Tempotron learning rule [11], ReSuMe [12], SWAT [13], Chronotron [14], and SPAN [15]. What is more, the neural engineering framework (NEF) was also used to train CNN for object detection with great success from the perspective of hardware implementation [16,17]. With these impressive training algorithms, the SNN has been successfully applied to perform simple tasks with small datasets. However, most of these algorithms are limited to shallow SNNs. The training algorithms for fully-connected SNN, convolutional multi-layer SNNs, or deep SNNs that are capable of implementing object recognition and object detection have also been developed extensively [18–24].

Recently, a gradient surrogate method was proposed to train deeper SNNs with multiple hidden layers [25–31]. With this algorithm, the spiking nonlinearity derivation was replaced by the derivation of a continuously differentiable function. However, the applications of SNN have been limited to some simple benchmark datasets. It is highly desirable to perform some engineering tasks to pave the practical applications of SNN.

Bridge damage detection, which is an important engineering task relating to the security of human beings, has attracted lots of attention in the fields of DL [32–37]. For example, crack damage detections have been successfully demonstrated with high accuracy based on convolutional neural network (CNN) [32,33], fully convolutional neural network [34], and faster region-based convolutional neural network [35,36]. However, the bridge damage detection based on a SNN, which is a promising solution to enable low power consumption, has not yet been reported.

In this paper, we propose a deep convolutional spiking neural network (DCSNN) to realize the bridge damage detection. Similar to the construction method of VGG network [38], we proposed a spiking VGG7 with leaky-integrate-and-fire (LIF) neurons. The dataset is manually collected from the actual concrete bridge. The rest of the paper is organized as follows. In Section 2, the network architecture of the DCSNN is described. The LIF model is also presented. Section 3 presents the surrogate gradient training algorithm. In Section 4, the training results are presented, and the inference process is performed. At last, concluding remarks are provided in Section 5.

2. Methodology

2.1. Network Structure of DCSNN

Inspired by the well-known VGG net, we developed a DCSNN named SpikingVGG7 as presented in Figure 1. SpikingVGG7 can be viewed as consisting of a spiking encoder, a spiking feature extractor, and a spiking classifier. The spiking encoder consists of the first convolution layer and the following spiking neuron layer, which are responsible for encoding the input image into a spiking sequence. The spiking feature extractor consists of four convolution layers, pooling layers, and spiking neuron layers together, and is responsible for feature extraction of spiking sequences output by spiking encoder. The spiking classifier consists of two fully connected layers and two spiking neuron layers, which are responsible for mapping the spiking feature sequence to the target space, and then obtain the final classification result through firing rate and softmax.

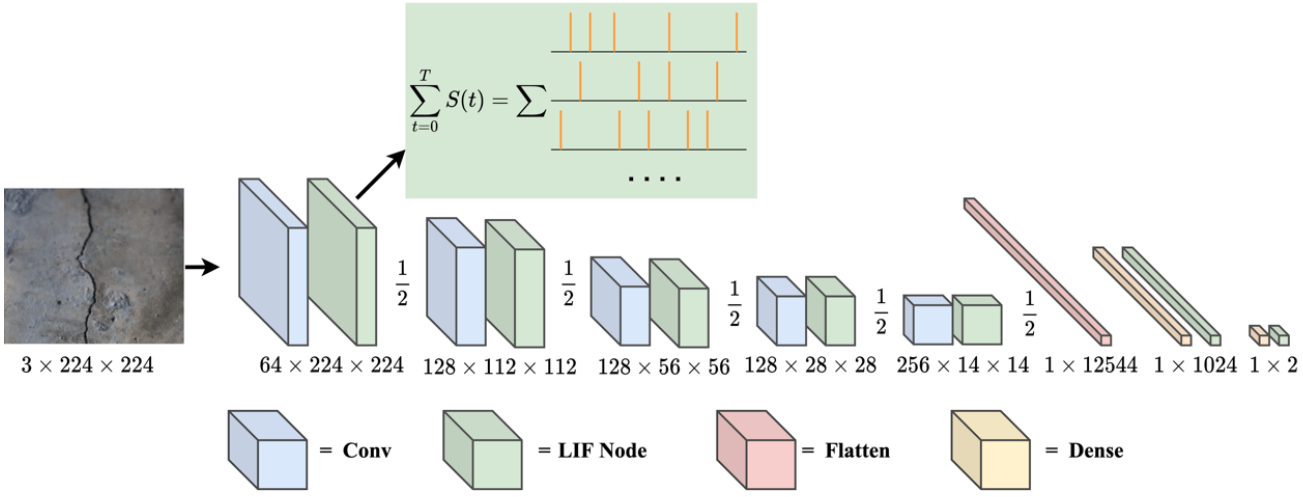

Figure 1. The architecture of the DCSNN for bridge damage detection.

In detail, in the SpikingVGG7 network, the size of the kernel of the convolution layer is 3×3. The padding method is used for convolution operation, and the feature map is fed into the spiking neuron, which will repeatedly calculate the output of convolution within the observation time T. The specific calculation method will be introduced later. The

spiking output from the spiking neuron will accumulate in the dimension of time and be fed into the maximum pooling layer, which is responsible for down-sampling the feature map on the plane. The size of the pooling layer is 2. After down-sampling, the length and width of the feature map will be reduced to 1/2 of the original input. After 5 times of 'convolution–spiking activation–pooling' processes, the obtained feature map will be flattened and stretched into one dimension, and then the obtained distributed feature map will be mapped to the sample space using the fully-connected layer, which is followed by spiking neurons. The output of a spiking neuron is binary, and the classified results of a single run may be easily disturbed. Therefore, it is generally considered that the output of the SNN is represented by the spike firing rate of the output layer within a period of time (observation time T). Therefore, the network needs to run for a period of time, that is, the average spike firing rate after T time is used as the classification basis. The network parameters are shown in Table 1.

Table 1. The network parameters for the DCSNN.

Layer (Type)	Output Shape	Parameter Number
Conv2d	[batch, 64, 224, 224]	1728
LIF Node	[T, batch, 64, 224, 224]	0
MaxPool2d	[batch, 128, 112, 112]	0
Conv2d	[batch, 128, 112, 112]	73,728
LIF Node	[T, batch, 128, 112, 112]	0
MaxPool2d	[batch, 128, 56, 56]	0
Conv2d	[batch, 128, 56, 56]	147,456
LIF Node	[T, batch, 128, 56, 56]	0
MaxPool2d	[batch, 128, 28, 28]	0
Conv2d	[batch, 256, 28, 28]	147,456
LIF Node	[T, batch, 256, 28, 28]	0
MaxPool2d	[batch, 256, 14, 14]	0
Conv2d	[batch, 256, 14, 14]	294,912
LIF Node	[T, batch, 256, 14, 14]	0
MaxPool2d	[batch, 256, 7, 7]	0
Flatten	[batch, 12544]	0
Linear	[batch, 1024]	12,845,056
LIF Node	[T, batch, 1024]	0
Linear	[batch, 2]	2048
LIF Node	[T, batch, 2]	0

2.2. LIF Model

Biological spiking neurons use spikes for transmission, communication, and calculation. There are various spiking neuron models such as the Hodgkin–Huxley (HH) neuron model, the integrate-and-fire (IF) neuron model, and the LIF neuron model. Here, classical LIF neurons are selected in the process of neuron modeling.

Mathematically, the LIF neuron can be modeled as follows,

$$\tau_m \frac{dU(t)}{dt} = -(U(t) - U_{reset}) + X(t) \tag{1}$$

where $\tau_m = 10$ is time constant, $U(t)$ is membrane voltage, U_{reset} is the reset voltage, $X(t)$ is the external stimulus.

A discrete difference equation is used to approximate the continuous differential equation as following,

$$\tau_m(U[t] - U[t-1]) = -(U[t-1] - U_{reset}) + X[t] \tag{2}$$

Therefore, the instantaneous membrane voltage $U[t]$ at time t can be obtained:

$$U[t] = f(U[t-1], X[t]) = U[t-1] + \frac{1}{\tau_m}(-(U[t-1] - U_{reset}) + X[t]) \tag{3}$$

In the following, $H[t]$ is used to represent the membrane voltage after the neuron is charged and before the spike is released, $V[t]$ is used to represent the membrane voltage after the neuron fires the spike, and $S[t]$ is used to represent the spike released by the neuron. Then, the discrete equations of charge and discharge stages are as follows:

$$[t] = f(V[t-1], X[t]) \tag{4}$$

$$S[t] = g(H[t] - V_{threshold}) \tag{5}$$

where $g(x)$ is a step function:

$$g(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases} \tag{6}$$

The hard reset is considered in experiments as:

$$V[t] = H[t] \cdot (1 - S[t]) + S[t] \cdot V_{reset} \tag{7}$$

For a constant external stimulus with $X(t) = 2$, the membrane voltage and the spike emission are presented in Figure 2. The member voltage is varied periodically even for a constant input. When the membrane voltage reaches the threshold, a spike is generated, and the membrane voltage is decreased sharply to the reset value.

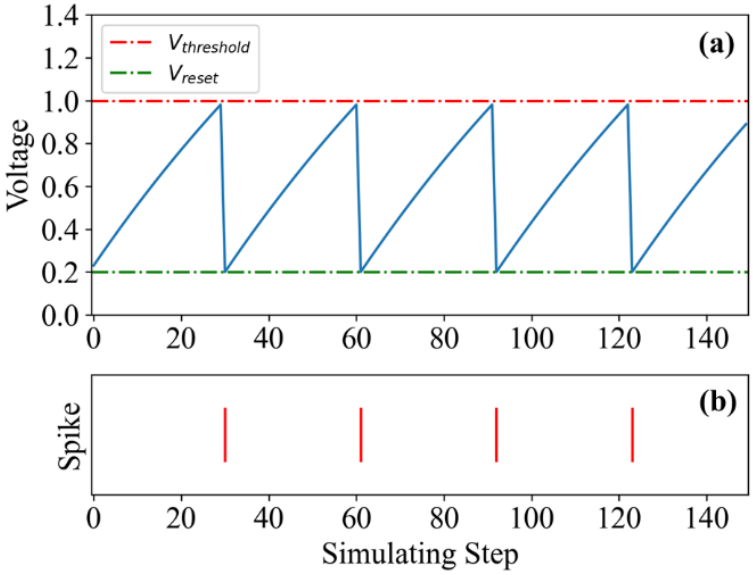

Figure 2. (**a**) The membrane voltage and (**b**) spike output of LIF neurons for X(t) = 2.

We consider 30 LIF neurons, and the external stimulus for each LIF neuron is increased linearly with the neuron index as shown in Figure 3a. The observation time step is T = 128. The membrane voltages and the corresponding spikes for the 30 LIF neurons at different time steps are presented in Figure 3b,c. By calculating the firing rate of each LIF neuron, it can be seen from Figure 3d that the firing rate is generally increased with the neuron index. The firing rate of an LIF neuron is used to replace the ReLU function.

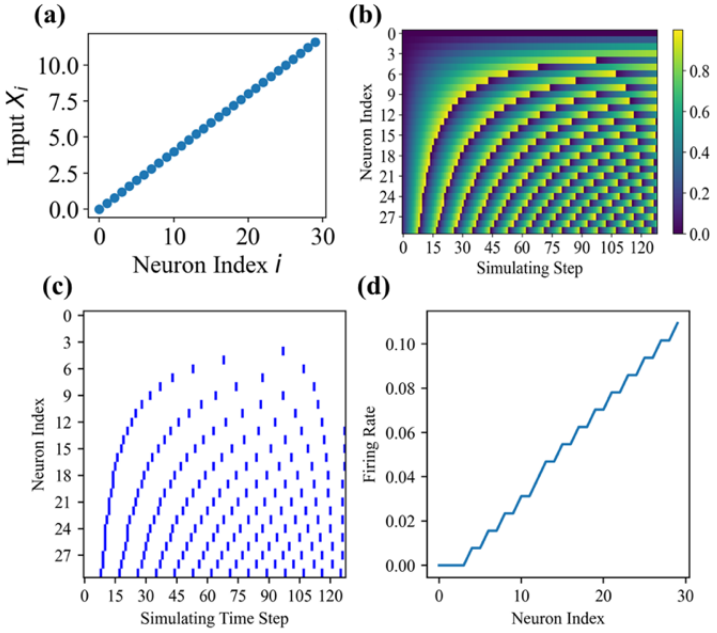

Figure 3. (**a**) External stimulus for each LIF neuron, (**b**) membrane voltage, and (**c**) spike emission for each LIF neuron at different time steps, (**d**) firing rate for each LIF neuron.

3. Training Algorithm

Note that, the SNN is difficult to train because of the non-differentiable spike output. It is not possible to train an SNN using back propagation normally, but it can be replaced with a similar shape but differentiable gating function, known as surrogate gradient [25]. Specifically, in forward propagation, step functions are used, and the output of the LIF neuron is discrete zeros and ones. For back propagation, the gradient is calculated using surrogate functions, such as sigmoid function.

Here, we adopt surrogate gradient learning to address the classification task for practical bridge damage dataset. In the present work, we consider four cases of surrogate functions as follows [39,40].

(i) Sigmoid function:

$$g(x) = sigmoid[\alpha x] = \frac{1}{1+e^{-\alpha x}} \tag{8}$$

(ii) arctan function:

$$g(x) = \frac{1}{\pi}\arctan\left(\frac{\pi}{2}\alpha x\right) + \frac{1}{2} \tag{9}$$

(iii) softsign function:

$$g(x) = \frac{1}{2}\left(\frac{\alpha x}{1+|\alpha x|} + 1\right) \tag{10}$$

(iv) erf function:

$$g(x) = \frac{1}{\sqrt{\pi}}\int_{-\infty}^{\alpha x} e^{-t^2} dt \tag{11}$$

For the four cases, we consider $\alpha = 1$. The normalized surrogate gradient functions and their corresponding gradients are presented in Figure 4.

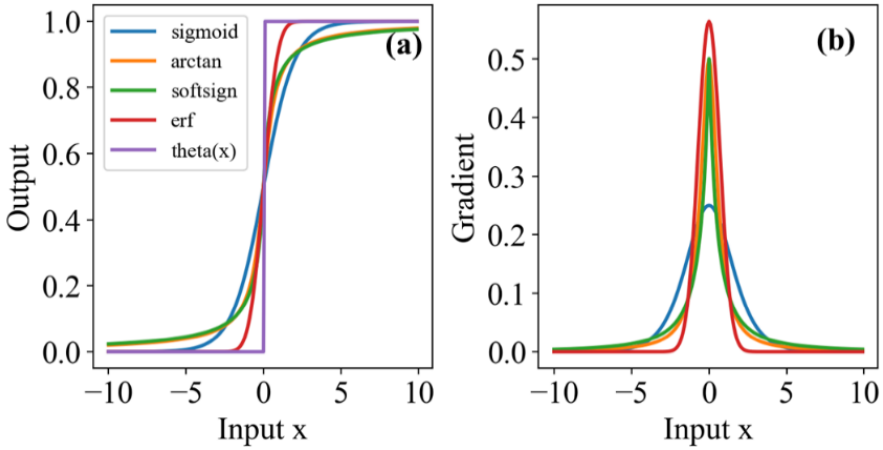

Figure 4. (**a**) Normalized surrogate gradient functions and (**b**) their corresponding gradients.

4. Experiments

4.1. Datasets

Our dataset is collected from the actual concrete bridge, and the original image resolution is 8868 × 4888. Since this study aims to classify damage, 9000 images with a resolution of 224 × 224 are selected and cropped from the original image. These images are divided into three categories. The area-type damage includes steel corrosion and concrete spalling. Note, in engineering, corrosion and concrete spalling usually appear together. The second type of damage is concrete crack damage. The third type is damage-less image. For the three types, each contains 3000 images. For each type, the training set contains 2700 images and the verification set contains 300 images.

It is worth mentioning that, in order to evaluate the performance of our model in practical application, we prepared another 500 images of each damage actually collected for the inference test. At the present stage, there is no similar public dataset, and part of crack data from the SDNET2018 dataset is selected for an inference test to verify the generalization performance [41]. Some representative samples in the dataset are presented in Figure 5.

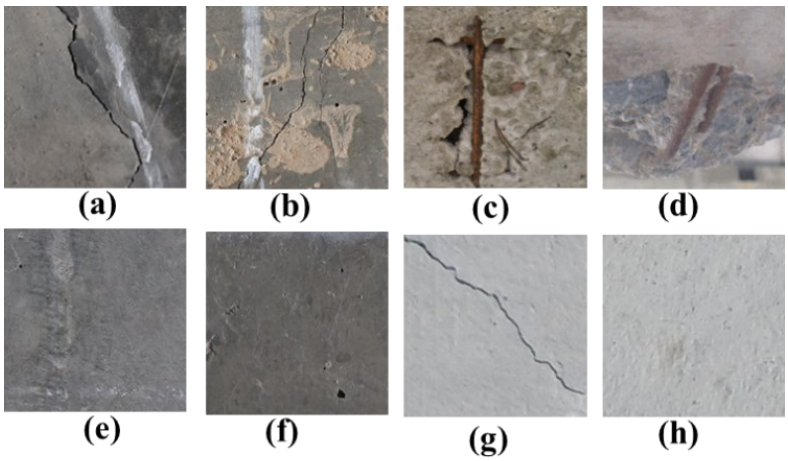

Figure 5. Some representative samples in the dataset. (**a**,**b**) correspond to the crack in our dataset, (**c**,**d**) correspond to the area-type damage in our dataset, (**e**,**f**) correspond to damage-less data, (**g**,**h**) denotes crack from SDNET 2018 [41].

4.2. Spike Encoding Strategy

The aim of spike encoding is to transform the image into spikes. Here, two spike encoding strategies are considered: the Poisson encoder and the convolution encoder.

Mathematically, the spike encoding for the Poisson encoder can be expressed as follows,

$$Spike[t] = g(random - x) \tag{12}$$

where a random function generates random floating numbers between 0 and 1, x is the normalized gray value of the input image.

Correspondingly, the spike encoding method for the convolution encoder can be expressed as spiking activation

$$Spike[t] = LIF(Conv2d(x)) \tag{13}$$

The *Conv2d* represents the convolution operation between the input x and the convolution kernel, and LIF means the spiking activation.

The Poisson code of the input image is accumulated in the time dimension. Figure 6b–e shows the visual effect of the accumulated spike encoding output when the observation time is T = 0, 7, 15, 31. It can be seen that, for a larger observation time T, the accumulated spike encoding output is closer to the original image. Thus, it can get more information about the original image. For the convolution encoder, the output feature size is [64, 224, 224]. Figure 6f–i correspond to four representative convolutional outputs. It is shown that, for a trained convolutional encoder, more different image features can be extracted.

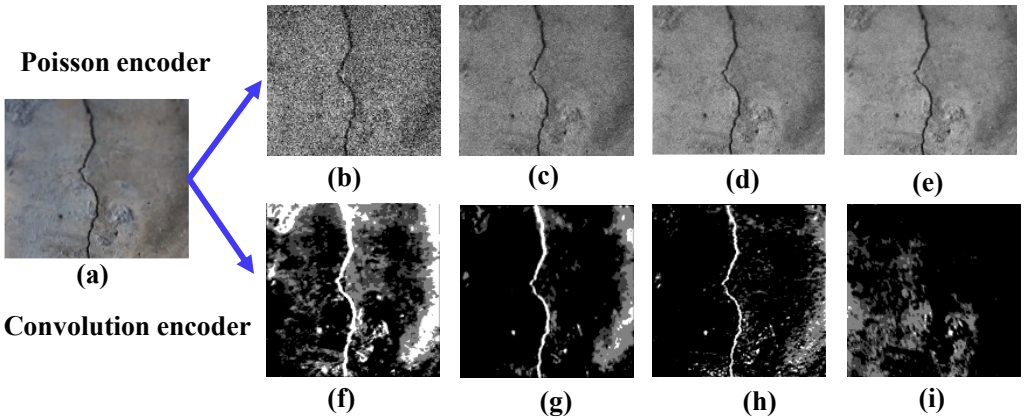

Figure 6. The spike encoding for the input image. (**a**) the original input image. (**b**–**e**) represents the accumulative output of the Poisson encoder at observation time T = 0, 7, 15, 31, respectively. (**f**–**i**) represents the cumulative feature map output of the convolutional encoder at T = 3, four channels were randomly selected from 64 channels.

4.3. Training

The experiments were performed on a computer with two Intel Xeon(R) E5-2620v4@2.1 GHz CPUs, 64 GB Random Access Memory, and NVIDIA GeForce RTX 2070 SUPER GPU. We adopt Pytorch and Spikingjelly DL framework [42], to build a VGG-like DCSNN. The batch size is 32, the learning rate is 0.01. The Adam optimizer is used here as the optimizing algorithm [43], and the loss function is the mean square error (MSE).

Here, the spike rate encoding is employed for the output results. The average spike rate during time T was used as the classification basis. The target result is represented by the neuron that fires at the highest frequency, and the rest neurons remain silent.

We adopt the surrogate gradient method to train the DCSNN in an end-to-end manner and compare classification performance for different gradient surrogate functions. Figure 7a,b, respectively, shows the verification accuracy and train loss of DCSNN and CNN in 300 training cycles. Four different gradient surrogate functions are considered in DCSNN. After training 300 states, each model is in a convergent state. The observation time is T = 6. It can be seen that $\arctan(x)$ leads to the best verification performance, while error function erf leads to the worst performance. Among the four gradient surrogate functions, arctan, $sigmoid$ erf, and $tanh$, the maximum verification accuracy in the training process is

99.0%, 98.1%, 95.8%, and 97.7%, respectively. Note, using the same network structure of ANN, the maximum accuracy is 99.6%.

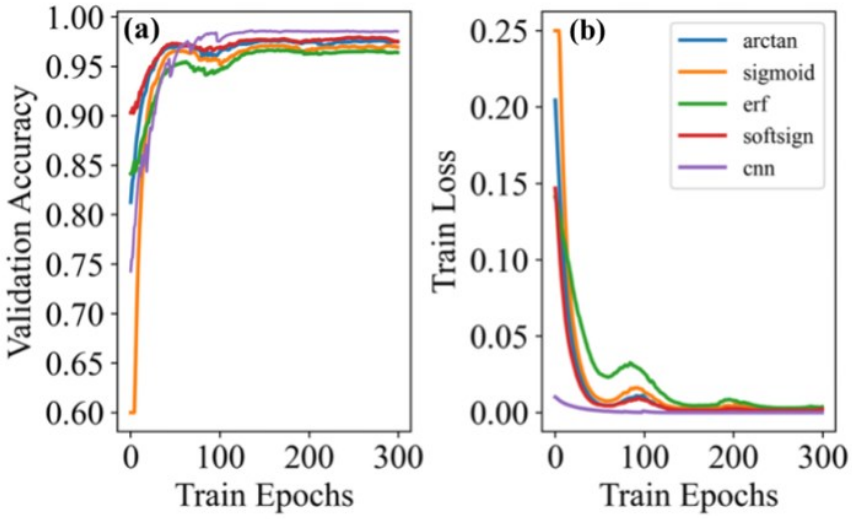

Figure 7. The validation accuracy (**a**) and training loss (**b**) for different gradient surrogate functions.

Next, the effect of observation time T on the validation performance is also considered. As shown in Figure 8a, with the increase of T, the classification accuracy increases firstly and saturates around 0.99 and then decreases. The convergence epoch is also presented in Figure 8b. Here, if the verification accuracy varies within 10% over 10 epochs, the first epoch is defined as the convergence epoch. It can be seen that when T is relatively small, the convergence epoch is relatively small, indicating that the network can converge rapidly, but the performance is poor due to insufficient features. As T increases, the convergence epoch is increased (i.e., the convergence speed is decreased) because the extracted features are more complex, but the accuracy is improved. The maximum accuracy is achieved at T = 6. Note, when T continues to increase, the model will be expanded from the time dimension into a very deep network, and the deep network has the risk of gradient disappearance and gradient explosion, resulting in the decline of the accuracy. As shown in Figure 8, when T = 16, the model classification accuracy drops to 61.0%. When T is greater than 16, the classifier loses its classification ability, and the model cannot learn.

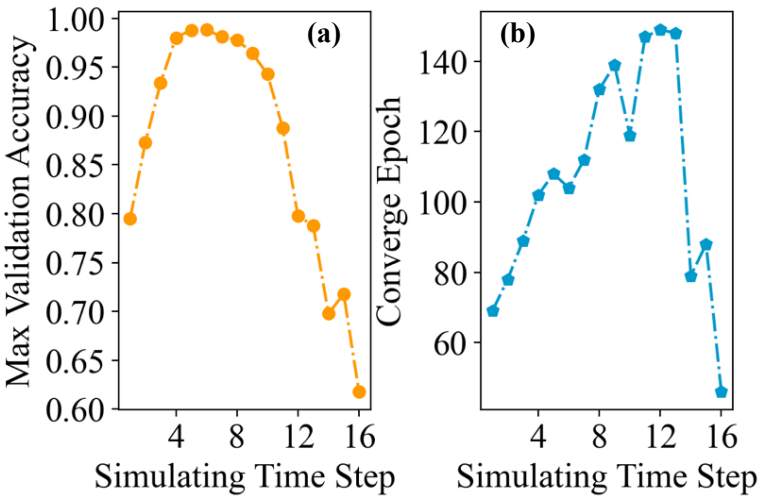

Figure 8. The effects of observation time T on the (**a**) validation accuracy and (**b**) convergence epoch.

4.4. Testing

In order to quantify the inference accuracy and generalization performance of the model, we used our own test set as well as some data from SDNET2018 dataset for testing [41]. The SDNET2018 is a dataset of cracks in concrete buildings, including 8484 crack images and 47,608 non-crack images. According to the shooting scenes, it can be divided into bridge deck, road surface, and wall surface. The images taken on the bridge crack are similar to our dataset, while the images taken in the other two scenes differ greatly. There are only two types of damage in our damage classification dataset, crack and area-type damage. Thus, we introduce 8484 crack images in SDNET2018 for testing.

Table 2 shows the testing performance of the model with 300 training cycles. Here, the observation time is T = 6. We can see that multiple models generally perform better on our own dataset. Compared with the model using convolution encoder and the model using Poisson encoder, the performance of SDNET2018 is quite different, reaching 78.45% and 70.47% classification accuracy, respectively. It can be seen that the convolution encoder presents better generalization performance because parameters can be learned, and more robust features are extracted in the process of coding. When the softsign is used as the surrogate gradient function, the convolution encoder performs better, and the accuracy can reach 97.83%. In addition, when the convolution encoder is used, the classification accuracy is 97.11%, 96.33%, and 97.67% for the cases with sigmoid, erf, and arctan as surrogate gradient function. In addition, we also tested on the concrete bridge damage dataset based on STDP-based unsupervised learning named SDNN [9], where the accuracy is 73.2% in our dataset and 55.3% in SDNET2018. Obviously, the performance of unsupervised learning is not high due to the lack of guidance of supervised signal.

Table 2. The testing accuracy for different training methods for two datasets.

Model	Accuracy on (%)	
	Ours	**SDNET2018**
VGG7	98.67	84.79
Spiking $VGG7_{T=6}$ + softsign + Convolutional Encode	97.83	78.45
Spiking $VGG7_{T=6}$ + softsign + Poisson Encode	91.22	70.47
Spiking $VGG7_{T=6}$ + sigmoid + Convolutional Encode	97.11	76.07
Spiking $VGG7_{T=6}$ + erf + Convolutional Encode	96.33	74.34
Spiking $VGG7_{T=6}$ + arctan + Convolutional Encode	97.67	76.68
SDNN	73.2	55.30

At last, we tested the bridge damage dataset using the ANN-SNN method. We converted the trained original VGG7 into an identically structured SNN network, which is called Converted VGG7. We tested the validation accuracy at different observation times T using the Converted VGG7 network, and the results are shown in Figure 9. Since we do not need to directly train the Converted VGG7 network, but only use the weights of the original VGG7 network, the observation time T can be set to a relatively large value when testing the accuracy with the Converted VGG7.

As can be seen from Figure 9, the validation accuracy increases with the increase of the observation time T, and finally remains basically unchanged. Especially, when t = 230, Converted VGG7 achieves the maximum classification accuracy of 96.8% in our dataset; when t = 256, Converted VGG7 achieved the maximum classification accuracy of 75% in SDNET2018. Compared to VGG7, SpikingVGG7 achieved a maximum validation accuracy of 97.83% in our dataset at T = 6, which is 1.03% higher than the maximum validation accuracy of Converted VGG7. Besides, SpikingVGG7 achieved a maximum accuracy of 78.45% in SDNET2018 at T = 6, which is 3.49% higher than the maximum validation accuracy of Converted VGG7. Therefore, it can be concluded that the SpikingVGG7 has a better classification effect than the Converted VGG7.

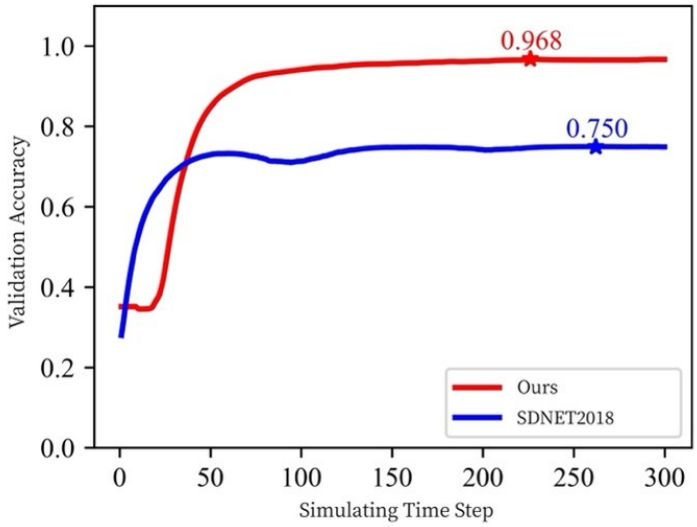

Figure 9. The validation accuracy of Converted VGG7 at different simulating time step.

4.5. Complexity Analysis

In the following, we also analyze the complexity of the SpikingVGG7 and the original one. The time complexity can be calculated as the number of operations performed on floating-point numbers. Considering the feedforward propagation, in the VGG7 net, the time complexity of the overall networks $\mathrm{O(VGG7)} = \sum_{l=1}^{D} M_l^2 \cdot K_l^2 \cdot C_{l-1} \cdot C_l$ [44]. D is the number of convolutional layers, l means the lth convolutional layer. C_l means the number of output channels of the lth convolutional layer. For the lth convolutional layer, the number of input channels is the number of output channels of the $(l-1)$th convolutional layer. M is the side length of feature map. K is the edge length of the convolution kernel. The network structure of the two networks is similar; however, SpikingVGG7 has an extra coding layer after each convolution layer and an extra time dimension T, hence the time complexity of SpikingVGG is $\mathrm{O(spiking\ VGG7)} = \left(\sum_{l=1}^{D} M_l^2 \cdot K_l^2 \cdot C_{l-1} \cdot C_l + M_l^2 \cdot Cl\right) \cdot T$. However, except for the first convolution layer based on 32-bit floating point number operations as in VGG7, in SpikingVGG, the calculation of the rest convolution layers is based on 0 and 1, which is much simpler than floating operations. Hence, it is difficult to compare time complexity of the two networks. As for space complexity, the network structures of SpikingVGG7 and VGG7 are similar; we consider that the space complexity is basically the same.

4.6. Hardware Implmentations

As the benefit of using SNNs is primarily evident when deployed on a neuromorphic computer, there are also works done for the hardware implementations with digital and analog approaches [45,46]. Here, we also briefly discuss the potential hardware implementations of the DCSNN with both digital and analog implementations.

As the basic element of a SNN, the LIF neuron could be implemented via CMOS circuits [47]; we firstly consider the hardware implementation based on analog circuits. CMOS-based adders and multipliers could be used for MAC operations in convolution, pooling layers and fully connected layers in the feed-forward propagation. The output of the fully-´connected layer could pass through an COMS-based integrator to count spikes. Then, a control module is required for the calculation of error and weight change. Voltage or current control signal can be sent to the multipliers for weight adaption.

For digital implementation, we use FPGA hardware circuits. In forward propagation, input data are sent into the data buffer, and the convolution operation is completed with the convolution kernel with shift registers according to the number of output feature graphs of each layer. In digital circuits, as the spikes are encoded with 0 or 1, only the adder is required to calculate the weighted sum of the input and the weight. At each clock cycle, the calculated membrane potential and a given threshold are compared through a comparator, and the

spike number could be calculated using a counter. Finally, the Softmax classifier completes the final output result after probability conversion by searching the corresponding value in ROM according to the input data. When back propagation, according to the label of ROM and comparing the calculated save to RAM, the calculated convolution kernel weight and bias values of updates will be updated by the controller.

5. Conclusions

We use the end-to-end training method to build and train a DCSNN that can classify concrete bridge damage in a real engineering environment. This is the first attempt of bridge damage detection using SNN. We verify experimentally that under the same conditions, with 300 training cycles, the classification accuracy can be achieved 97.83% by using the softsign function as the gradient surrogate function. Note, such accuracy is very close to the performance of the CNN (98.67%) with the same network structure. Although SNN is not completely comparable to ANN in performance, it makes more sense in the following two aspects: 1. It simulates spike signals transmitted in biological neural networks and attempts to reveal the mechanism of information processing; 2. Binary spike coding used in SNN can effectively reduce computational complexity and enhance power efficiency. In addition, we have examined the effect of observation time step T on the network performance. The experimental results show that with the increase of the observation time T, both the model precision and the convergence speed increase first and then show a downward trend. The maximum accuracy is achieved at T = 6. In the comparison experiment of encoders, in our test set, the two encoders lead to the best accuracy with 97.83% and 91.22%. Note, there was a significant difference in performance when SDNET2018 was used as test data. The DCSNN using gradient surrogate method can achieve performance with 78.45% and 70.47% for two encoders. Thus, the test results on SDNET2018 show that CNN exhibits have better generalization performance. As a further attempt, we will continue to study the realization of DCSNN in other more complex visual tasks.

Author Contributions: Conceptualization, S.X.; methodology, S.X. and S.J.; software, S.J. and X.L.; validation, T.Z and L.Y.; formal analysis, T.Z. and L.Y.; data curation, S.J.; writing—original draft preparation, S.X. and S.J.; writing—review and editing, X.L.; visualization, X.L.; supervision, S.X.; funding acquisition, S.X. All authors have read and agreed to the published version of the manuscript.

Funding: National Key Research and Development Program of China (2021YFB2801900, 2021YFB2801901, 2021YFB2801902, 2021YFB2801904); National Natural Science Foundation of China (No. 61974177, No. 61674119); National Outstanding Youth Science Fund Project of National Natural Science Foundation of China (62022062); The Fundamental Research Funds for the Central Universities (JB210114).

Data Availability Statement: Data are available upon request.

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

References

1. Maass, W. Networks of spiking neurons: The third generation of neural network models. *Neural Netw.* **1997**, *10*, 1659–1671. [CrossRef]
2. Wang, X.; Lin, X.; Dang, X. Supervised learning in spiking neural networks: A review of algorithms and evaluations. *Neural Netw.* **2020**, *125*, 258–280. [CrossRef] [PubMed]
3. Taherkhani, A.; Belatreche, A.; Li, Y.; Cosma, G.; Maguire, L.; McGinnity, T. A review of learning in biologically plausible spiking neural networks. *Neural Netw.* **2020**, *122*, 253–272. [CrossRef] [PubMed]
4. Caporale, N.; Dan, Y. Spike timing—Dependent plasticity: A Hebbian learning rule. *Annu. Rev. Neurosci.* **2008**, *31*, 25–46. [CrossRef]
5. Diehl, P.; Cook, M. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* **2015**, *9*, 99. [CrossRef]
6. Xiang, S.; Zhang, Y.; Gong, J. STDP-based unsupervised spike pattern learning in a photonic spiking neural network with VCSELs and VCSOAs. *IEEE J. Sel. Top. Quantum Electron.* **2019**, *25*, 1–9. [CrossRef]
7. Xiang, S.; Ren, Z.; Song, Z. Computing primitive of fully VCSEL-based all-optical spiking neural network for supervised learning and pattern classification. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 2494–2505. [CrossRef] [PubMed]

8. Song, Z.; Xiang, S.; Cao, X.; Zhao, S.; Hao, Y. Experimental demonstration of photonic spike-timing dependent plasticity based on a VCSOA. *Sci. China Inf. Sci.* **2022**, *65*, 182401.
9. Kheradpisheh, S.; Ganjtabesh, M.; Thorpe, S. STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* **2018**, *99*, 56–67. [CrossRef]
10. Bohte, S.; Kok, J.; La Poutre, H. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* **2002**, *48*, 17–37. [CrossRef]
11. Gütig, R.; Sompolinsky, H. The tempotron: A neuron that learns spike timing—Based decisions. *Nat. Neurosci.* **2006**, *9*, 420–428. [CrossRef] [PubMed]
12. Ponulak, F.; Kasi'nski, A. Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting. *Neural Comput.* **2010**, *22*, 467–510. [CrossRef] [PubMed]
13. Wade, J.; McDaid, L.; Santos, J.; Sayers, H. SWAT: A spiking neural network training algorithm for classification problems. *IEEE Trans. Neural Netw.* **2010**, *21*, 1817–1830. [CrossRef] [PubMed]
14. Florian, R. The chronotron: A neuron that learns to fire temporally precise spike patterns. *PLoS ONE* **2012**, *7*, e40233. [CrossRef]
15. Mohemmed, A.; Schliebs, S.; Matsuda, S.; Kasabov, N. SPAN: Spike pattern association neuron for learning spatio-temporal spike patterns. *Int. J. Neural Syst.* **2012**, *22*, 1250012. [CrossRef] [PubMed]
16. Eliasmith, C.; Anderson, C. *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*; MIT press: Cambridge, MA, USA, 2003.
17. Tsur, E. *Neuromorphic Engineering: The Scientist's, Algorithm Designer's, and Computer Architect's Perspectives on Brain-Inspired Computing*; CRC Press: Boca Raton, FL, USA, 2021.
18. Sporea, I.; Grüning, A. Supervised learning in multilayer spiking neural networks. *Neural Comput.* **2013**, *25*, 473–509. [CrossRef]
19. Cao, Y.; Chen, Y.; Khosla, D. Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* **2015**, *113*, 54–66. [CrossRef]
20. Lee, J.; Delbruck, T.; Pfeiffer, M. Training deep spiking neural networks using backpropagation. *Front. Comput. Neurosci.* **2016**, *10*, 508. [CrossRef]
21. Lin, X.; Wang, X.; Hao, Z. Supervised learning in multilayer spiking neural networks with inner products of spike trains. *Neurocomputing* **2017**, *237*, 59–70. [CrossRef]
22. Yamazaki, K.; Vo-Ho, V.-K.; Bulsara, D.; Le, N. Spiking neural networks and their applications: A Review. *Brain Sci.* **2022**, *12*, 863. [CrossRef]
23. Taherkhani, A.; Belatreche, A.; Li, Y.; Maguire, L. A supervised learning algorithm for learning precise timing of multiple spikes in multilayer spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 5394–5407. [CrossRef] [PubMed]
24. Kim, S.; Park, S.; Na, B.; Yoon, S. Spiking-YOLO: Spiking neural network for energy-efficient object detection. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020.
25. Neftci, E.; Mostafa, H.; Zenke, F. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Process. Mag.* **2019**, *36*, 51–63. [CrossRef]
26. Qiao, G.; Ning, N.; Zuo, Y.; Hu, S.; Yu, Q.; Liu, Y. Direct training of hardware-friendly weight binarized spiking neural network with surrogate gradient learning towards spatio-temporal event-based dynamic data recognition. *Neurocomputing* **2021**, *457*, 203–213. [CrossRef]
27. Zenke, F.; Ganguli, S. SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks. *Neural Comput.* **2018**, *30*, 1514–1541. [CrossRef]
28. Shrestha, S.; Orchard, G. SLAYER: Spike Layer Error Reassignment in Time. In Proceedings of the Advances in Neural Information Processing Systems, Montréal, QC, Canada, 3–8 December 2018; pp. 1419–1428.
29. Wu, Y.; Deng, L.; Li, G.; Zhu, J.; Xie, Y.; Shi, L. Direct training for spiking neural networks: Faster, larger, better. In Proceedings of the AAAI Conference on Artificial Intelligence, Hawaii, NA, USA, 27 January–1 February 2019; Volume 33, pp. 1311–1318.
30. Wu, J.; Chua, Y.; Zhang, M.; Li, G.; Li, H.; Tan, K. A Tandem Learning Rule for Effective Training and Rapid Inference of Deep Spiking Neural Networks. *arXiv* **2020**, arXiv:1907.01167. [CrossRef]
31. Deng, L. Rethinking the performance comparison between SNNS and ANNS. *Neural Netw.* **2020**, *121*, 294–307. [CrossRef]
32. Cha, Y.; Choi, W.; Büyüköztürk, O. Deep learning-based crack damage detection using convolutional neural networks. *Comput.-Aided Civ. Infrastruct. Eng.* **2017**, *32*, 361–378. [CrossRef]
33. Chen, F.; Jahanshahi, M. NB-CNN: Deep learning-based crack detection using convolutional neural network and Naïve Bayes data fusion. *IEEE Trans. Ind. Electron.* **2017**, *65*, 4392–4400. [CrossRef]
34. Dung, C. Autonomous concrete crack detection using deep fully convolutional neural network. *Autom. Constr.* **2019**, *99*, 52–58. [CrossRef]
35. Deng, J.; Lu, Y.; Lee, V. Concrete crack detection with handwriting script interferences using faster region-based convolutional neural network. *Comput.-Aided Civ. Infrastruct. Eng.* **2020**, *35*, 373–388. [CrossRef]
36. Yu, L.; He, S.; Liu, X. Engineering-oriented bridge multiple-damage detection with damage integrity using modified faster region-based convolutional neural network. *Multimed. Tools Appl.* **2022**, *81*, 18279–18304. [CrossRef]
37. Yu, L.; He, S.; Liu, X. Intelligent crack detection and quantification in the concrete bridge: A deep learning-assisted image processing approach. *Adv. Civ. Eng.* **2022**, *2022*, 1813821. [CrossRef]

38. Sengupta, A.; Ye, Y.; Wang, R.; Liu, C.; Roy, K. Going Deeper in Spiking Neural Networks: VGG and Residual Architectures. *Front. Neurosci.* **2019**, *13*, 95. [CrossRef] [PubMed]
39. Wu, Y.; Deng, L.; Li, G.; Zhu, J.; Shi, L. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Front. Neurosci.* **2018**, *12*, 331. [CrossRef]
40. Yin, S.; Venkataramanaiah, S.; Chen, G.; Krishnamurthy, R.; Cao, Y.; Chakrabarti, C.; Seo, J. Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations. In Proceedings of the 2017 IEEE Biomedical Circuits and Systems Conference (BioCAS), Torino, Italy, 19–21 October 2017.
41. Dorafshan, S.; Thomas, R.; Maguire, M. SDNET2018: An annotated image dataset for non-contact concrete crack detection using deep convolutional neural networks. *Data Brief* **2018**, *21*, 1664–1668. [CrossRef]
42. Github. Available online: https://github.com/fangwei123456/spikingjelly (accessed on 17 December 2019).
43. Kingma, D.; Ba, J. Adam: A Method for Stochastic Optimization. Available online: https://arxiv.org/abs/1412.6980/ (accessed on 12 December 2014).
44. He, K.; Sun, J. Convolutional neural networks at constrained time cost. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Santiago, Chile, 7–13 December 2015; pp. 5353–5360.
45. Davies, M.; Narayan, S.; Tsung-Han, L. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* **2018**, *38*, 82–99. [CrossRef]
46. Hazan, A.; Ezra, E. Neuromorphic Neural Engineering Framework-Inspired Online Continuous Learning with Analog Circuitry. *Appl. Sci.* **2022**, *12*, 4528. [CrossRef]
47. Kornijcuk, V.; Lim, H.; Seok, J. Leaky integrate-and-fire neuron circuit based on floating-gate integrator. *Front. Neuro-Sci.* **2016**, *10*, 212. [CrossRef]