machine learning & knowledge extraction

MDPI

Article

Improving Spiking Neural Network Performance with Auxiliary Learning

Paolo G. Cachi [1,†], Sebastián Ventura [2] and Krzysztof J. Cios [1,*]

[1] Department of Computer Science, Virginia Commonwealth University, Richomnd, VA 23220, USA; pcachi@vcu.edu
[2] Department of Computing and Artificial Intelligence, Universidad de Córdoba, 14071 Córdoba, Spain; sventura@uco.es
* Correspondence: kcios@vcu.edu
† Current address: 601 West Main Street, Richmond, VA 23220, USA.

Abstract: The use of back propagation through the time learning rule enabled the supervised training of deep spiking neural networks to process temporal neuromorphic data. However, their performance is still below non-spiking neural networks. Previous work pointed out that one of the main causes is the limited number of neuromorphic data currently available, which are also difficult to generate. With the goal of overcoming this problem, we explore the usage of auxiliary learning as a means of helping spiking neural networks to identify more general features. Tests are performed on neuromorphic DVS-CIFAR10 and DVS128-Gesture datasets. The results indicate that training with auxiliary learning tasks improves their accuracy, albeit slightly. Different scenarios, including manual and automatic combination losses using implicit differentiation, are explored to analyze the usage of auxiliary tasks.

Keywords: auxiliary learning; spiking neural networks; implicit differentiation

Citation: Cachi, P.G.; Ventura, S.; Cios, K.J. Improving Spiking Neural Network Performance with Auxiliary Learning. *Mach. Learn. Knowl. Extr.* **2023**, *5*, 1010–1022. https://doi.org/10.3390/make5030052

Academic Editor: Andreas Holzinger

Received: 22 July 2023
Revised: 2 August 2023
Accepted: 3 August 2023
Published: 5 August 2023

1. Introduction

Spiking neural networks (SNNs) have great potential for low-powered artificial intelligence applications when they are implemented on specialized hardware [1–3]. In contrast to non-spiking neural networks (ANNs) [4–6] that process information by successive non-linear transformations, SNNs use dynamic units, called spiking neurons, for handling spatiotemporal data. The SNN use discrete "spike" events to represent the activation of neurons. This makes them more efficient as the communication through spikes is a sparse process. When implemented on neuromorphic hardware, such as Intel's Loihi2 neuromorphic chip [7,8], they use orders of magnitude less energy than ANNs. This energy efficiency makes them attractive for various applications, such as sign gesture recognition [9,10], heartbeat signal classification [11], continual learning in 3D scenarios [12], and optimization [13].

In practice, however, the usage of SNNs for solving real-life problems is still limited because of their performance. A key issue linked with their lower performance is the limited size of available temporal data for their training [14,15]. As dynamic systems, SNNs require and are better suited for processing temporal data, such as neuromorphic data [16]. Unfortunately, there is a small number of temporal datasets currently available, and what makes it even worse is that they often have a small number of instances. As a result, SNNs trained with such datasets often suffer from over-fitting and unstable convergence. In order to improve their efficiency on limited-size data, other techniques need to be found.

The problems with training on small-size data are not specific to SNN but are present in machine learning in general [17,18]. Two methods have been proposed to address this problem: data augmentation [17,18] (the creation of new synthetic data by the modification of input samples or latent feature vectors) and the use of regularization methods [19–21]

(direct regularization by penalty loss or indirect regularization with auxiliary learning). Only input data augmentation has been studied within the framework of SNN [22].

In this paper, we study the use of auxiliary learning as an indirect regularization method for SNN training. Auxiliary learning was used for improving the performance of ANNs. Works such as [19,20,23] explored the use of one or more secondary tasks as a way of regularization. These attempts have been helpful in increasing their performance. The limitations seen in the ANN framework are still present in SNNs. In this paper, we explore the selection of the auxiliary tasks as well as the relation of their number to be used.

The specific contributions of our paper are as follows:

- We utilized AL for training SNN and experimentally demonstrated that using one or more auxiliary tasks increases the performance.
- We performed an analysis of different auxiliary task learning setups and analyzed their influence on performance.
- We compared the results with state-of-the-art SNN solutions and showed that using AL improves their accuracy.

The rest of the paper is structured as follows. Section 2 presents a detailed overview of the related work. Section 3 describes the proposed framework for SNN auxiliary learning. Section 4 describes the experimental settings and results. The paper ends with conclusions.

2. Related Work

2.1. Spiking Neural Networks

SNNs are designed to emulate the way that biological neurons generate and transmit electrical impulses, known as spikes, to other neurons [1–3]. This allows for processing information in an event-driven and temporally precise way, which is more efficient and well suited for solving a variety of tasks. Training SNNs requires specialized algorithms that can deal with the temporal processing of spiking neurons in the presence of non-linearity associated with spiking neuron communication. Noteworthy works providing an overview of training methodologies for SNNs include [25**?** –27].

There are several different approaches to training SNNs that depend on a specific task to be solved and on the learning rule used for updating the weights. Some of the most used learning rules include spike-timing-dependent plasticity (STDP) for unsupervised local learning [28–30], reward modulated STDP for reinforcement learning through STDP [31,32], and spike-based backpropagation (SBP) [33–35] and remote supervised learning (ReSuMe) [36,37] for supervised learning. Here, we use the SBP algorithm because of its advantages over the other training methods, including enhanced efficiency and the capability to leverage standard optimization techniques developed for deep neural network architectures.

SBP is a variant of the backpropagation algorithm commonly used to train ANNs. Like the standard backpropagation algorithm, SBP uses gradient descent to update the weights in order to minimize the error between the predicted and desired output. However, unlike the standard backpropagation algorithm, SBP is designed to handle the dynamic operation of spiking neurons and the associated non-linearity of the communication via spikes. To enable the incorporation of temporal dependencies in the training process, SBP uses backpropagation through time (BPTT) [35], and in order to overcome the non-linearity of the spiking mechanisms, it uses surrogate gradient functions (SGD) [34].

BPTT was originally developed for training recurrent neural networks, which can process sequences of inputs [38]. In BPTT, the error signal is propagated back through the network not just over a single time step but over multiple time steps. This allows the network to take into account the past history of inputs and outputs while adjusting its weights. The trick that allows the use of BPTT in SNNs is that the spiking neuron model can be unfolded into a recurrent computation system [35].

Although implementing the computational graph of the spiking operation is feasible through tools such as PyTorch or TensorFlow, the non-differentiability of the Heaviside function impedes the correct calculation of the error gradients. To overcome this issue, the SGD method is used [34]. The main idea behind SGD is to use a replacement function as

the gradient for the non-differentiable function. This replacement is only used during the gradient calculation or backward pass training stage. The Heaviside function is still used during the forward pass to maintain the correct operation of the network.

2.2. Auxiliary Learning

Auxiliary learning (AL) is a technique developed to improve the performance of ANN when training data are limited in size or expensive to collect [19–21]. In auxiliary learning, a model is trained on multiple tasks in a similar manner as used in multitask learning (MTL), see Figure 1. The difference between MTL and AL is that while MTL strives for good performance on all tasks (all tasks being of equal importance), AL focuses on the performance of one main task that the network needs to solve and treats all the other tasks as auxiliary (used only to help improve the performance of the main task). The auxiliary tasks can be related or not to the main task.

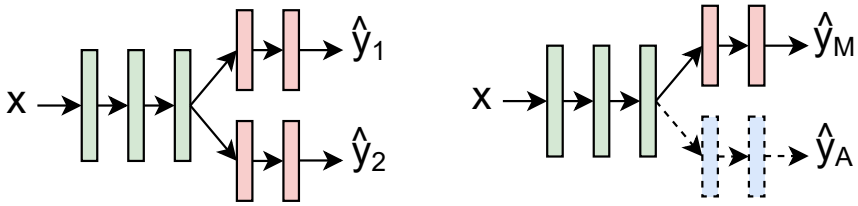


Figure 1. (**Left**): In multitask learning, the goal is to perform more than one learning task at the same time, with all tasks being equally important. (**Right**): In auxiliary learning, the goal is to learn one main task while using one or more auxiliary tasks.

AL approach has several advantages. Training the network on multiple tasks simultaneously using AL forces it to learn more general, transferable to other tasks features. This improves performance on the main task. AL can also improve the efficiency of training as the network can learn from the auxiliary tasks without the need for additional training data. This makes it practical for training complex neural networks. Finally, AL can serve as a regularization tool for network learning, which usually improves its generalization ability by reducing overfitting.

Learning multiple tasks, however, creates problems such as negative transfer [39], i.e., when different tasks have conflicting goals, like increasing performance for one task decreases the performance of the other task(s). Another challenge is knowing how to efficiently combine multiple loss functions, i.e., how to weight the losses so the main task is preferred [40]. In this paper, we address these questions by exploring different setups for loss error combination and the number of auxiliary tasks required. We also compare the efficiency of AL in training SNN vs. training ANN (also with AL).

2.3. Input Data Augmentation

Data augmentation is a technique used to increase the size and diversity of a dataset [17,18]. In the input data augmentation, additional data are generated by applying various transformations to the original input data, such as rotation, scaling, cropping, or adding noise. Doing this provides more examples to learn from and can help the trained model to generalize better on new data. Researchers studied the use of geometrical transformations for input data augmentation on neuromorphic data for training SNN. Using this approach allowed for about a 4% accuracy increase [14]. This illustrates one of the problems of SNN, namely, the scarcity of event-based data for their training. In this paper, in addition to input data augmentation, we use AL as a method to increase accuracy on limited size training data.

3. Methods

3.1. Problem Definition

Consider an input space X, where $X \in \mathbb{R}^n$, and a main task T_{main} and one or more auxiliary tasks $T_{aux}^{(i)}$. The expected output for the main task is Y_{main} and for the auxiliary tasks Y_{aux}^{i}. We want to train a spiking neural network, $f(x)$, with weights W that minimize the loss of T_{main} while using $T_{aux}^{(i)}$ as a regularization method during training. Note that $T_{aux}^{(i)}$ is used during training only.

3.2. Architecture

The auxiliary learning architecture for training SNNs is shown in Figure 2. It consists of a feature extraction block connected in a feed-forward fashion to the main task and auxiliary task(s) blocks. The spiking input signal is processed by the first block, the feature extraction block, into a latent p-dimensional spiking feature vector, which is then fed to the main and auxiliary task classifier blocks to find the outputs. The idea behind this architecture is to allow the feature extraction block to receive feedback from the main classifier block (main task loss) and also from the auxiliary task classifier block(s) (auxiliary task losses) during training. In this way, the auxiliary task classifier blocks act as regularization blocks for the feature extraction block.

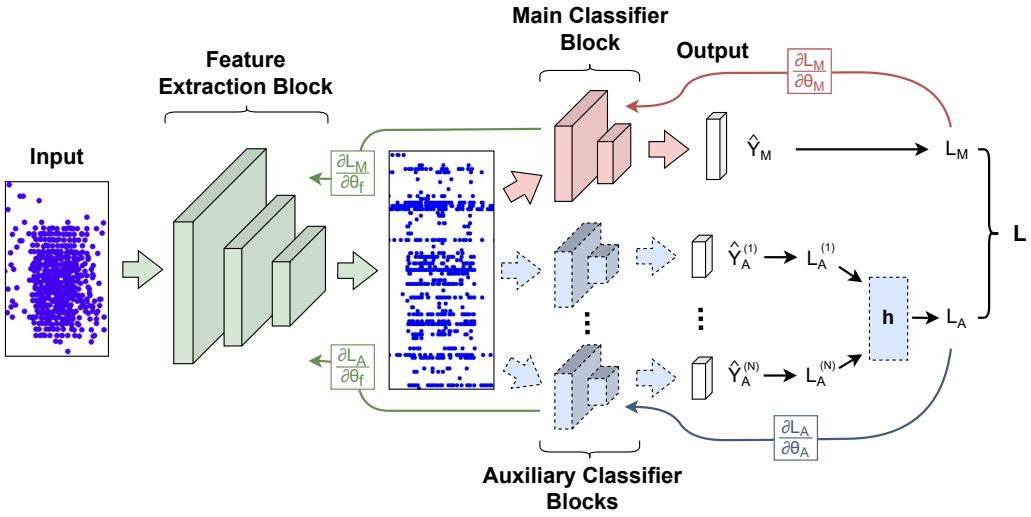


Figure 2. Auxiliary learning architecture. The network uses a multitask architecture in which only one task, "the main task", is of importance. The other tasks, "the auxiliary tasks", are used as additional regularization losses for helping the main task performance. The auxiliary tasks are only used during training.

In this work as the spiking neuron model, we use the parametric leaky integrate and fire neuron model (PLIF) [22], which is a leaky integrate and fire (LIF) [41,42] neuron with learnable time constants. The equations for the membrane potential, $U_i^{(l)}[n]$, and synaptic current, $I_i^{(l)}[n]$, of neuron i in layer l are given by:

$$U_i^{(l)}[n+1] = \alpha U_i^{(l)}[n] + I_i^{(l)}[n] - S_i^{(l)}[n] \tag{1}$$

$$I_i^{(l)}[n+1] = \beta I_i^{(l)}[n] + \sum_j W_{ij}^{(l)} S_j^{(l)}[n] \tag{2}$$

where α and β are decay constants equal to $\alpha \equiv exp(-\frac{\triangle_t}{\tau_{mem}})$ and $\beta \equiv exp(-\frac{\triangle_t}{\tau_{syn}})$ with a small simulation time step $\triangle_t > 0$ and membrane and synaptic time constants τ_{mem} and τ_{syn}; W_{ij} are synaptic weights of the postsynaptic neuron i and presynaptic neurons j and j

within the same layer l; and $S_i^{(l)}[n]$ is the output spike train of neuron i in layer l at time step $[n]$. The output spike train is expressed as the Heaviside step function of the difference between the membrane voltage and the firing threshold φ as follows:

$$S_j^{(l)}[n] = \Theta(U_j^{(l)}[n] - \varphi) \tag{3}$$

3.3. Training and Testing

The goal is to learn a set of weights, W^*, that minimizes the loss of the main task while utilizing the auxiliary losses as regularization parameters. This can be expressed as the following optimization problem:

$$W^* = \underset{W}{\operatorname{argmin}}\, L \tag{4}$$

where L represents the total loss, which is calculated from the main task loss, L_M, and the auxiliary task losses, $L_A^{(i)}$, as follows:

$$L = L_M + h(L_A^{(1)}, L_A^{(2)}, \ldots, L_A^{(i)}) \tag{5}$$

where L_M is the main task loss; L_{aux}^i are the auxiliary task losses; and h is a linear/non-linear operation that processes the auxiliary losses. The simplest loss combination case is when h(.) is a linear combination of the auxiliary losses. In this scenario, the total loss, L, can be expressed as:

$$L = (1 - \alpha) * L_M + \alpha * \sum_{i=0}^{N} \gamma_i L_A^{(i)} \tag{6}$$

where α is a loss rate constant that controls the rate between the main and auxiliary losses, and γ_i denotes weights assigned to each auxiliary loss (they can be determined through manual tuning methods like grid search, or automatic tuning methods like implicit differentiation [43]). The latter approach can also be used to train function $h(.)$ when a non-linear model is chosen. In this work, we compare the results obtained by all three methods: the manual tuning of a linear combination, the automatic tuning of a linear model, and the automatic tuning of a non-linear model for $h(.)$.

During testing, the samples are only fed into the feature extraction block and then into the main task classifier block. The auxiliary task blocks are not used since the focus is solely on evaluating the performance of the main task.

3.4. Implementation

The network is implemented using the SpikingJelly framework, which consists of a set of python libraries for supervised training of SNNs [44]. The code for the network implementation as well as all sof the experiments and results is posted at GitHub (https://github.com/PaoloGCD/AuxiliaryLearning-SNN (accessed on 22 July 2023)).

4. Experiments and Results

We evaluate effectiveness of AL in SNN for solving recognition tasks using the CIFAR10-DVS [45] and DVS128-Gesture [46] neuromorphic datasets. All of the tests are performed using the architecture shown in Figure 2. For structuring the network, we followed the architecture used in [14,22], which is based on the VGG structure [47]. Specifically, the number of layers used for the feature extraction and classifier blocks for each dataset is shown in Table 1. Each layer of the feature extraction block is composed of PLIF neurons in a convolutional layer with batch normalization that is followed by max pooling with kernel 2×2. All convolution operations use a kernel size of 3×3 with stride 1 and padding 1. The number of channels for all convolution layers is 128. The layers of the classifier blocks (the main and auxiliary) are composed of a fully connected layer of PLIF neurons with a dropout of 0.5. The number of features of the first fully connected layer is set to 1/4 of the

number of input vector features. The number of features for the output layer (the last fully connected layer) is 10 times the number of classes of the average voting when stride 10 is used for computing the classification label. All of the results are presented as the average of ten runs.

Table 1. Network architecture used for analyzing DVS-CIFAR10 and DVS128-Gesture neuromorphic data.

Dataset	Number of Layers per Block	
	Feature Extraction	Main/Auxiliary Classifier
DVS-CIFAR10	4	2
DVS128-Gesture	5	2

4.1. Training with One Auxiliary Task

First, we test the performance of training SNN with just one auxiliary task. For each dataset, we test three different auxiliary task configurations. The labels used for the main (M) and auxiliary (A) tasks are shown in Table 2.

Table 2. The main task (M) and auxiliary task (A1, A2, and A3) configurations.

CIFAR10-DVS					DVS128-Gesture				
Class	**M**	**A1**	**A2**	**A3**	**Class**	**M**	**A1**	**A2**	**A3**
Airplane	0	0	0	0	Hand clapping	0	0	1	0
Automobile	1	1	0	1	Right hand wave	1	1	3	1
Bird	2	2	1	2	Left hand wave	2	2	2	2
Cat	3	3	1	3	Right arm clockwise	3	3	3	1
Deer	4	4	1	4	Right arm counter clock	4	4	3	1
Dog	5	5	1	3	Left arm clockwise	5	5	2	2
Frog	6	6	1	5	Left arm counter clock	6	6	2	2
Horse	7	7	1	4	Arm roll	7	7	0	0
Ship	8	8	0	0	Air drums	8	8	0	0
Truck	9	9	0	1	Air guitar	9	9	4	3
					Other gestures	10	10	5	4

For DVS-CIFAR10 data, A1 is selected as a duplicate of the main task label; A2 is categorization into living vs. non-living class labels; and A3 is based on morphological properties of the classes. For example, deer and horses are put into the same group (group 4) because of their morphological similarities. For DVS128-Gesture data, A1 is again a duplicate of the main task, while A2 and A3 are two different categorization tasks based on the morphological properties of the images. For example, hand clapping, arm rolling, and air drums are one auxiliary category (see Table 2, column A3) as they show the usage of both hands.

For the above cases only a linear combination loss (Equation (6)) is used. We test different values of the loss rate constant α. Specifically, we use α values 0.1, 0.2, 0.3, 0.4, and 0.5. Table 3 shows the validation accuracy after 250 training epochs for the two datasets while using different auxiliary tasks and loss rate constants. The validation dataset was randomly selected as a separate part from the training dataset; its size is equal to 10% of the training dataset. We chose 250 training epochs based on the observation that at around this number the network's accuracy reaches a plateau with no significant improvement (see Figure 3, which illustrates validation accuracy over 1024 training epochs on CIFAR10-DVS and DVSGesture128 datasets).

Both data augmentation and auxiliary learning improve the accuracy of the SNN. Data augmentation results in a more significant increase in performance, while the utilization of auxiliary learning further improves the performance achieved through data augmentation

alone. It is worth noting that there is a decline in performance when using task A2 for CIFAR10-DVS data. This decrease can be attributed to the fact that auxiliary tasks should find useful information to facilitate learning. Apparently A2 does not provide such information for the network since living vs. non-living categorization is based on a very abstract concept that the network is not able to handle.

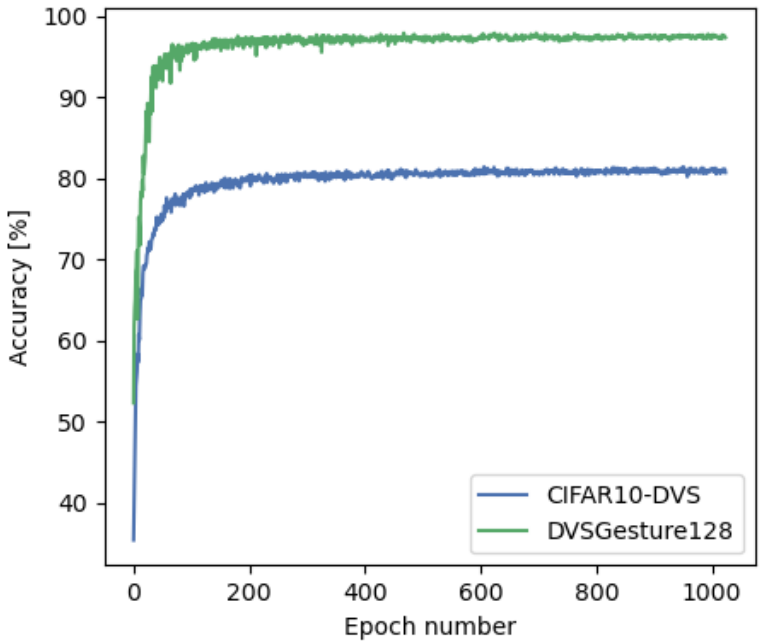


Figure 3. Validation accuracy on CIFAR10-DVS and DVSGesture128 datasets.

Table 3. Validation accuracy on DVS-CIFAR10 and DVS128-Gesture datasets using auxiliary learning for 250 training epochs. Bold values represent the best result within the column.

Model	Validation Accuracy after 250 Epochs (%)					
	CIFAR10-DVS			DVSGesture128		
	A1	A2	A3	A1	A2	A3
ST-SNN	72.24 ± 0.35	72.24 ± 0.35	72.24 ± 0.35	96.07 ± 0.27	96.07 ± 0.27	96.07 ± 0.27
ST-SNN + aug	80.83 ± 0.70	80.83 ± 0.70	80.83 ± 0.70	98.32 ± 0.31	98.32 ± 0.31	98.32 ± 0.31
AL-SNN + aug + $\alpha = 0.1$	80.98 ± 0.38	**81.00 ± 0.40**	80.78 ± 0.31	98.50 ± 0.33	98.55 ± 0.37	98.44 ± 0.28
AL-SNN + aug + $\alpha = 0.2$	81.60 ± 0.55	80.35 ± 0.71	81.02 ± 0.67	98.50 ± 0.26	98.50 ± 0.33	98.61 ± 0.28
AL-SNN + aug + $\alpha = 0.3$	81.38 ± 0.47	79.45 ± 0.70	**81.13 ± 0.58**	98.67 ± 0.37	**98.73 ± 0.26**	**98.61 ± 0.17**
AL-SNN + aug + $\alpha = 0.4$	81.00 ± 0.49	78.90 ± 0.39	81.05 ± 0.42	98.44 ± 0.33	98.61 ± 0.20	98.32 ± 0.13
AL-SNN + aug + $\alpha = 0.5$	**81.75 ± 0.33**	78.72 ± 0.44	80.85 ± 0.81	**98.67 ± 0.13**	98.38 ± 0.16	98.55 ± 0.31

Regarding the choice of the loss rate constant, higher values (greater than 0.3) yield better results (except for case A2 for CIFAR10-DVS data). However, the difference in performance is not clear-cut, making the manual selection of this parameter quite challenging. Because of this, we use an automated method for selecting the loss rate constant; it is described below in Section 4.3.

4.2. Training with More Than One Auxiliary Task

Table 4 shows the testing accuracies of AL with two (AL-SNN-2T), three (AL-SNN-3T), and four (AL-SNN-4T) auxiliary tasks. The first three auxiliary tasks are the same classification tasks as in Table 2. The fourth auxiliary task is randomly generated as a four-label classification. For convenience of comparison, the results for ST-SNN and the

best results for AL-SNN trained with one auxiliary task (repeated from Table 3) are also shown.

Table 4. Validation accuracy on DVS-CIFAR10 and DVS128-Gesture datasets using multiple auxiliary tasks for 250 training epochs. Bold values represent the best result within the column.

Model	Validation Accuracy after 250 Epochs (%)	
	CIFAR10-DVS	**DVSGesture128**
ST-SNN	72.24 ± 0.35	96.07 ± 0.48
ST-SNN + aug	80.83 ± 0.70	98.32 ± 0.31
AL-SNN + aug	**81.75 ± 0.33**	**98.73 ± 0.26**
AL-SNN-2T + aug	80.22 ± 0.64	98.38 ± 0.31
AL-SNN-3T + aug	80.67 ± 0.24	98.67 ± 0.24
AL-SNN-4T + aug	80.77 ± 0.54	98.73 ± 0.33

Observe that training with more auxiliary tasks did not yield better results than using a single auxiliary task. The process of determining the appropriate selection of auxiliary tasks, determining their respective weights, and choosing a proper combination of loss rate becomes highly challenging, rendering a manual grid search unfeasible. In our test, the uniform combination of weights of 1 for all auxiliary losses and a loss rate of 0.5 is not the optimal choice, as seen from the results. Given the complexities involved in the manual combination of multiple auxiliary tasks, an automated method for combining them becomes essential to effectively leverage their strength, which is described next.

4.3. Using Implicit Differentiation

Here, we use implicit differentiation to train a loss combination function, h, such that L is minimized (Equation (5)). Table 5 shows the testing accuracies of training AL using all four auxiliary tasks and implicit differentiation. h is tested for both linear (AL-SNN-IDL-4T) and non-linear (AL-SNN-IDNL-4T) cases. Traditional ANN with three hidden layers is used for the non-linear case.

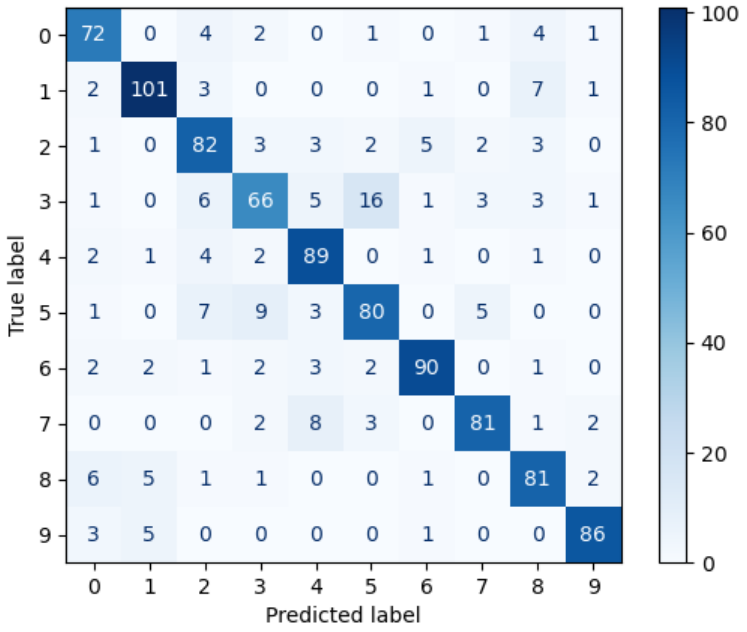
Table 5. Validation accuracy on DVS128-Gesture datasets using implicit differentiation for 250 training epochs. Bold values represent the best result within the column.

Model	Validation Accuracy after 250 Epochs (%)	
	CIFAR10-DVS	**DVSGesture128**
ST-SNN	72.24 ± 0.35	96.07 ± 0.48
ST-SNN + aug	80.83 ± 0.70	98.32 ± 0.31
AL-SNN + aug	**81.75 ± 0.33**	98.73 ± 0.26
AL-SNN-IDL-4T + aug	81.15 ± 0.27	98.67 ± 0.24
AL-SNN-IDNL-4T + aug	81.69 ± 0.34	**98.84 ± 0.39**

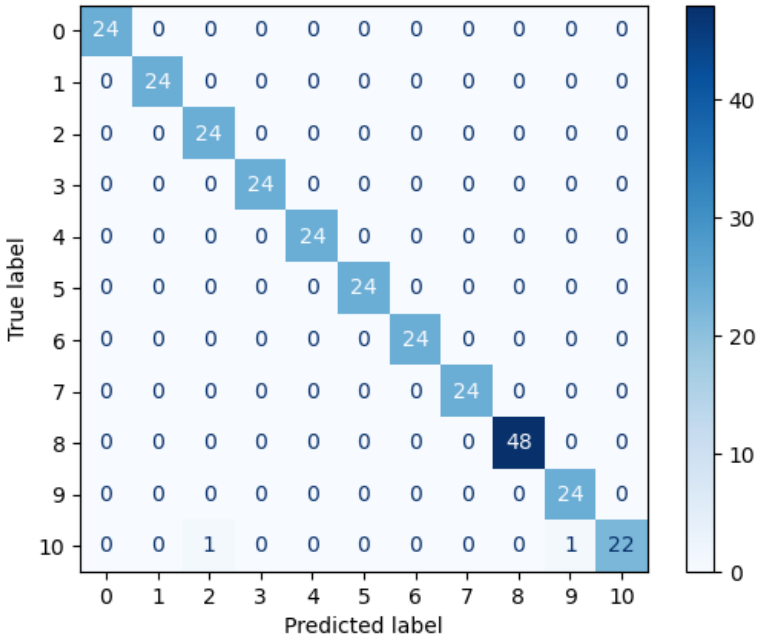
Observe that employing automatic differentiation with a non-linear function h yields the best overall result. When a linear function h is used, the obtained result is very close to the best outcome achieved through a manual grid search. These findings show that automatic differentiation not only mitigates the challenges associated with manual grid search but also improves the SNN system performance. It is important to highlight that A4 is a random task that does not provide any useful information, yet automatic differentiation successfully handles this task. This underscores the robustness and adaptability of automatic differentiation in effectively handling diverse tasks, even when they apparently do not provide additional information.

4.4. Comparison with State-of-the-Art SNNs

The proposed training approach using auxiliary learning with state-of- the-art methods is compared with using SNN on the CIFAR10-DVS and DVSGesture128 neuromorphic datasets. To identify the best trained networks, we conduct an analysis using precision, recall, and the F1-score. We then select the top-performing network for each dataset. Figure 4 shows the confusion matrix for the selected networks, and Table 6 shows the above performance indicators. The results are shown for 1024 training epochs on the testing set.



(**a**) CIFAR10-DVS.



(**b**) DVSGesture-128.

Figure 4. Confusion matrix for best performing SNN with AL for CIFAR10-DVS (**a**) and DVSGesture128 (**b**) datasets.

Table 6. Testing accuracy, precision, recall, and F1 score for best performing SNN with AL for CIFAR10-DVS and DVSGesture128 datasets.

Dataset	Model	Accuracy	Precision	Recall	F1-Score
CIFAR10-DVS	AL-SNN + aug + $\alpha = 0.5$	82.80	0.829	0.828	0.827
DVSGesture128	AL-SNN-IDNL-4T + aug	99.31	0.993	0.993	0.993

Overall, SNN trained using auxiliary learning exhibits a well-balanced performance in predicting the labels for each dataset. It is worth highlighting a particular case, which is the prediction of class 3 (cat) for CIFAR10-DVS data. This specific class is the most challenging to predict in the CIFAR10-DVS dataset.

We compare the obtained results with state-of-the-art SNNs, which are shown in Table 7.

Table 7. Comparison with state-of-the-art SNNs for CIFAR10-DVS and DVSGesture-128 datasets. Bold values represent the best result within the column.

Model	Reference	CIFAR10-DVS	DVSGesture-128
STBP [48]	AAAI 2021	67.80	96.87
PLIF [22]	ICCV 2021	74.80	97.57
Dspike [49]	NeurIPS 2021	75.40	-
AutoSNN [50]	ICML 2022	72.50	96.53
RecDis [51]	CVPR 2022	72.42	-
DSR [52]	CVPR 2022	77.27	-
NDA [14]	ECCV 2022	81.70	-
SpikeFormer [53]	ICLR 2023	80.90	98.30
AIA [54]	ICASSP 2023	**83.90**	-
AL-SNN (ours)	-	82.80	**99.31**

Notice that training with auxiliary learning achieves the highest accuracy for the DVSGesture128 dataset and the second highest for CIFAR10-DVS. The highest accuracy for CIFAR10-DVS is achieved by AIA, which is an SNN that uses a more advanced neuron model than the PLIF neuron model used in this work. In fact, we see that training with AL achieves higher accuracy when compared with SNN, which uses PLIF neurons (PLIF and NDA). We expect that AL with the AIA neuron model would achieve the best performance.

5. Conclusions

In this paper, we present the usage of auxiliary learning, in addition to data augmentation, to improve the performance of SNNs. The used network architecture consists of a feature extraction block connected in a feedforward fashion to a main classification block and one or more auxiliary task classification blocks. By using auxiliary tasks, we use additional information during training that helps in the regularization of the feature extraction block. As a result, the feature extraction block is forced to learn more general and robust features, which helps improve the SNN network performance on the main task. The results confirm that using AL during training indeed results in improved performance. Moreover, the experiments demonstrate that the extent of the improvement depends on the careful tuning combination of the loss rate parameters. To overcome this challenge, we use automatic differentiation [43] to automatically adjust the loss combination parameters. Note that all the experiments presented in this study were conducted through simulation using the SpikingJelly neuromorphic library. However, in the future we plan to leverage Intel's Lava framework, which enables one to directly deploy the network on the Loihi2 neuromorphic chip.

Author Contributions: P.G.C., S.V. and K.J.C. contributed to conception and design of the study. P.G.C. organized the database, performed the statistical analysis, and wrote the first draft of the manuscript. P.G.C. and K.J.C. wrote sections of the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The code for the network implementation as well as all the experiments and results is posted at https://github.com/PaoloGCD/AuxiliaryLearning-SNN (accesed on 22 July 2023).

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

Abbreviations

The following abbreviations are used in this manuscript:

AL	Auxiliary learning
ANNs	Artificial neural networks
BPTT	Backpropagation through time
LIF	Leaky integrate and fire
MTL	Multitask learning
PLIF	Parametric leaky integrate and fire
SBP	Spike-based backpropagation
SGD	Stochastic gradient descent
SNNs	Spiking neural networks
STDP	Spike-timing-dependent plasticity

References

1. Ghosh-Dastidar, S.; Adeli, H. Spiking Neural Networks. *Int. J. Neural Syst.* **2009**, *19*, 295–308. [CrossRef] [PubMed]
2. Gerstner, W.; Kistler, W.M. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*; Cambridge University Press: Cambridge, UK, 2002. [CrossRef]
3. Maass, W. Networks of spiking neurons: The third generation of neural network models. *Neural Netw.* **1997**, *10*, 1659–1671. [CrossRef]
4. Alzubaidi, L.; Zhang, J.; Humaidi, A.J.; Al-Dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaría, J.; Fadhel, M.A.; Al-Amidie, M.; Farhan, L. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *J. Big Data* **2021**, *8*, 53. [CrossRef] [PubMed]
5. Emmert-Streib, F.; Yang, Z.; Feng, H.; Tripathi, S.; Dehmer, M. An Introductory Review of Deep Learning for Prediction Models with Big Data. *Front. Artif. Intell.* **2020**, *3*, 4. [CrossRef]
6. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
7. Orchard, G.; Frady, E.P.; Rubin, D.B.D.; Sanborn, S.; Shrestha, S.B.; Sommer, F.T.; Davies, M. Efficient Neuromorphic Signal Processing with Loihi 2. In Proceedings of the 2021 IEEE Workshop on Signal Processing Systems (SiPS), Coimbra, Portugal, 19–21 October 2021.
8. Davies, M.; Wild, A.; Orchard, G.; Sandamirskaya, Y.; Guerra, G.A.F.; Joshi, P.; Plank, P.; Risbud, S.R. Advancing Neuromorphic Computing with Loihi: A Survey of Results and Outlook. *Proc. IEEE* **2021**, *109*, 911–934. [CrossRef]
9. Mohammadi, M.; Chandarana, P.; Seekings, J.; Hendrix, S.; Zand, R. Static hand gesture recognition for American sign language using neuromorphic hardware. *Neuromorphic Comput. Eng.* **2022**, *2*, 044005. [CrossRef]
10. Ceolini, E.; Frenkel, C.; Shrestha, S.B.; Taverni, G.; Khacef, L.; Payvand, M.; Donati, E. Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing. *Front. Neurosci.* **2020**, *14*, 637. [CrossRef]
11. Buettner, K.; George, A.D. Heartbeat Classification with Spiking Neural Networks on the Loihi Neuromorphic Processor. In Proceedings of the 2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Tampa, FL, USA, 7–9 July 2021; pp. 138–143. [CrossRef]
12. Hajizada, E.; Berggold, P.; Iacono, M.; Glover, A.; Sandamirskaya, Y. Interactive Continual Learning for Robots: A Neuromorphic Approach. In Proceedings of the International Conference on Neuromorphic Systems, Knoxville, TN, USA, 27–29 July 2022; Association for Computing Machinery: New York, NY, USA, 2022. [CrossRef]
13. Smith, J.D.; Severa, W.; Hill, A.J.; Reeder, L.; Franke, B.; Lehoucq, R.B.; Parekh, O.D.; Aimone, J.B. Solving a Steady-State PDE Using Spiking Networks and Neuromorphic Hardware. In Proceedings of the International Conference on Neuromorphic Systems, Oak Ridge, TN, USA, 28–30 July 2020; Association for Computing Machinery: New York, NY, USA, 2020. [CrossRef]
14. Li, Y.; Kim, Y.; Park, H.; Geller, T.; Panda, P. Neuromorphic Data Augmentation for Training Spiking Neural Networks. In Proceedings of the Computer Vision—ECCV 2022: 17th European Conference, Tel Aviv, Israel, 23–27 October 2022; Avidan, S., Brostow, G., Cissé, M., Farinella, G.M., Hassner, T., Eds.; Springer Nature: Cham, Switzerland, 2022; pp. 631–649.

15. Yin, B.; Corradi, F.; Bohté, S.M. Effective and Efficient Computation with Multiple-timescale Spiking Recurrent Neural Networks. In Proceedings of the International Conference on Neuromorphic Systems, Oak Ridge, TN, USA, 28–30 July 2020. [CrossRef]
16. Kugele, A.; Pfeil, T.; Pfeiffer, M.; Chicca, E. Efficient Processing of Spatio-Temporal Data Streams With Spiking Neural Networks. *Front. Neurosci.* **2020**, *14*, 439. [CrossRef]
17. Khalifa, N.E.; Loey, M.; Mirjalili, S. A comprehensive survey of recent trends in deep learning for digital images augmentation. *Artif. Intell. Rev.* **2022**, *55*, 2351–2377. [CrossRef] [PubMed]
18. Shorten, C.; Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. *J. Big Data* **2019**, *6*, 60. [CrossRef]
19. Shi, B.; Hoffman, J.; Saenko, K.; Darrell, T.; Xu, H. Auxiliary Task Reweighting for Minimum-data Learning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 7148–7160. [CrossRef]
20. Liu, S.; Davison, A.; Johns, E. Self-Supervised Generalisation with Meta Auxiliary Learning. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., Eds.; Curran Associates, Inc.: New York, NY, USA, 2019; Volume 32.
21. Du, Y.; Czarnecki, W.M.; Jayakumar, S.M.; Farajtabar, M.; Pascanu, R.; Lakshminarayanan, B. Adapting Auxiliary Losses Using Gradient Similarity. *arXiv* **2018**, arXiv:1812.02224.
22. Fang, W.; Yu, Z.; Chen, Y.; Masquelier, T.; Huang, T.; Tian, Y. Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seattle, WA, USA, 13–19 June 2020. [CrossRef]
23. Schröder, F.; Biemann, C. Estimating the influence of auxiliary tasks for multi-task learning of sequence tagging tasks. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Virtual Event, 5–10 July 2020; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 2971–2985. [CrossRef]
24. Training Spiking Neural Networks Using Lessons From Deep Learning. *arXiv* **2021**, arXiv:2109.12894.
25. Tavanaei, A.; Ghodrati, M.; Kheradpisheh, S.R.; Masquelier, T.; Maida, A. Deep learning in spiking neural networks. *Neural Netw.* **2019**, *111*, 47–63. [CrossRef]
26. 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 Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, Honolulu, HW, USA, 27 January–1 February 2019; AAAI Press: Washington, DC, USA, 2019. [CrossRef]
27. Lee, J.H.; Delbruck, T.; Pfeiffer, M. Training Deep Spiking Neural Networks Using Backpropagation. *Front. Neurosci.* **2016**, *10*, 508. [CrossRef] [PubMed]
28. Gerstner, W.; Kistler, W.M. Mathematical formulations of Hebbian learning. *Biol. Cybern.* **2002**, *87*, 404–415. [CrossRef] [PubMed]
29. Hebb, D.O. *The Organization of Behavior: A Neuropsychological Theory*; Wiley: Hoboken, NJ, USA, 1949.
30. Konorski, J. *Conditioned Reflexes and Neuron Organization*; Cambridge University Press: Cambridge, UK, 1948.
31. Frémaux, N.; Sprekeler, H.; Gerstner, W. Functional requirements for reward-modulated spike-timing-dependent plasticity. *J. Neurosci.* **2010**, *30*, 13326–13337. [CrossRef]
32. Legenstein, R.; Pecevski, D.; Maass, W. A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback. *PLoS Comput. Biol.* **2008**, *4*, 1–27. [CrossRef] [PubMed]
33. Kaiser, J.; Mostafa, H.; Neftci, E. Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE). *Front. Neurosci.* **2020**, *14*, 424. [CrossRef] [PubMed]
34. Neftci, E.O.; 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]
35. Shrestha, S.B.; Orchard, G. SLAYER: Spike Layer Error Reassignment in Time. *arXiv* **2018**, arXiv:1810.08646.
36. Ponulak, F.; Kasiński, A. Supervised Learning in Spiking Neural Networks with ReSuMe: Sequence Learning, Classification, and Spike Shifting. *Neural Comput.* **2010**, *22*, 467–510. [CrossRef] [PubMed]
37. Nski, A.; Ponulak, F.F. Comparison of supervised learning methods for spike time coding in spiking neural networks. *Int. J. Appl. Math. Comput. Sci.* **2006**, *16*, 101–113.
38. Mozer, M.C. A Focused Backpropagation Algorithm for Temporal Pattern Recognition. *Complex Syst.* **1989**, *3*, 348–381.
39. Wang, Z.; Dai, Z.; Póczos, B.; Carbonell, J. Characterizing and Avoiding Negative Transfer. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018. [CrossRef]
40. Standley, T.; Zamir, A.R.; Chen, D.; Guibas, L.; Malik, J.; Savarese, S. Which Tasks Should Be Learned Together in Multi-task Learning? In Proceedings of the 37th International Conference on Machine Learning, Virtual Event, 12–18 July 2020; Volume 119, pp. 9120–9132.
41. Gerstner, W.; Kistler, W.M.; Naud, R.; Paninski, L. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*; Cambridge University Press: New York, NY, USA, 2014.
42. Kandel, E.R.; Schwartz, J.H.; Jessell, T.M.; of Biochemistry, D.; Jessell, M.B.T.; Siegelbaum, S.; Hudspeth, A. *Principles of Neural Science*; McGraw-hill: New York, NY, USA, 2013; Volume 5.
43. Navon, A.; Achituve, I.; Maron, H.; Chechik, G.; Fetaya, E. Auxiliary Learning by Implicit Differentiation. *arXiv* **2020**, arXiv:2007.02693.
44. Fang, W.; Chen, Y.; Ding, J.; Chen, D.; Yu, Z.; Zhou, H.; Masquelier, T.; Tian, Y. SpikingJelly. 2020. Available online: https://github.com/fangwei123456/spikingjelly (accessed on 8 July 2023).

45. Li, H.; Liu, H.; Ji, X.; Li, G.; Shi, L. CIFAR10-DVS: An Event-Stream Dataset for Object Classification. *Front. Neurosci.* **2017**, *11*, 309. [CrossRef]
46. Amir, A.; Taba, B.; Berg, D.; Melano, T.; McKinstry, J.; Di Nolfo, C.; Nayak, T.; Andreopoulos, A.; Garreau, G.; Mendoza, M.; et al. A Low Power, Fully Event-Based Gesture Recognition System. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 7388–7397. [CrossRef]
47. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2014**, arXiv:1409.1556.
48. Zheng, H.; Wu, Y.; Deng, L.; Hu, Y.; Li, G. Going Deeper With Directly-Trained Larger Spiking Neural Networks. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020.
49. Li, Y.; Guo, Y.; Zhang, S.; Deng, S.; Hai, Y.; Gu, S. Differentiable Spike: Rethinking Gradient-Descent for Training Spiking Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems, Virtual Event, 6–14 December, 2021; Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2021; Volume 34, pp. 23426–23439.
50. Na, B.; Mok, J.; Park, S.; Lee, D.; Choe, H.; Yoon, S. AutoSNN: Towards Energy-Efficient Spiking Neural Networks. In Proceedings of the International Conference on Machine Learning, Baltimore, MD, USA, 17–23 July 2022.
51. Guo, Y.; Tong, X.; Chen, Y.; Zhang, L.; Liu, X.; Ma, Z.; Huang, X. RecDis-SNN: Rectifying Membrane Potential Distribution for Directly Training Spiking Neural Networks. In Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 18–24 June 2022; pp. 326–335. [CrossRef]
52. Meng, Q.; Xiao, M.; Yan, S.; Wang, Y.; Lin, Z.; Luo, Z.Q. Training High-Performance Low-Latency Spiking Neural Networks by Differentiation on Spike Representation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vancouver, BC, Canada, 18–24 June 2023.
53. Zhou, Z.; Zhu, Y.; He, C.; Wang, Y.; YAN, S.; Tian, Y.; Yuan, L. Spikformer: When Spiking Neural Network Meets Transformer. In Proceedings of the Eleventh International Conference on Learning Representations, Kigali, Rwanda, 1–5 May 2023.
54. Shen, H.; Luo, Y.; Cao, X.; Zhang, L.; Xiao, J.; Wang, T. Training Stronger Spiking Neural Networks with Biomimetic Adaptive Internal Association Neurons. In Proceedings of the ICASSP 2023—2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Rhodes Island, Greece, 4–10 June 2023; pp. 1–5. [CrossRef]