entropy

Article

Learning in Convolutional Neural Networks Accelerated by Transfer Entropy

Adrian Moldovan [1,2], Angel Caţaron [1,2,*] and Răzvan Andonie [1,3]

[1] Department of Electronics and Computers, Transilvania University, 500024 Braşov, Romania; adrian.moldovan@siemens.com (A.M.); razvan.andonie@cwu.edu (R.A.)
[2] Technology, Siemens SRL, 500007 Braşov, Romania
[3] Department of Computer Science, Central Washington University, Ellensburg, WA 98926, USA
* Correspondence: cataron@unitbv.ro; Tel.: +40-268-413000

Abstract: Recently, there is a growing interest in applying Transfer Entropy (TE) in quantifying the effective connectivity between artificial neurons. In a feedforward network, the TE can be used to quantify the relationships between neuron output pairs located in different layers. Our focus is on how to include the TE in the learning mechanisms of a Convolutional Neural Network (CNN) architecture. We introduce a novel training mechanism for CNN architectures which integrates the TE feedback connections. Adding the TE feedback parameter accelerates the training process, as fewer epochs are needed. On the flip side, it adds computational overhead to each epoch. According to our experiments on CNN classifiers, to achieve a reasonable computational overhead–accuracy trade-off, it is efficient to consider only the inter-neural information transfer of the neuron pairs between the last two fully connected layers. The TE acts as a smoothing factor, generating stability and becoming active only periodically, not after processing each input sample. Therefore, we can consider the TE is in our model a slowly changing meta-parameter.

Keywords: transfer entropy; causality; Convolutional Neural Network; deep learning

check for updates

Citation: Moldovan, A.; Caţaron, A.; Andonie, R. Learning in Convolutional Neural Networks Accelerated by Transfer Entropy. *Entropy* **2021**, *23*, 1218. https://doi.org/10.3390/e23091218

Academic Editor: Friedhelm Schwenker

Received: 14 August 2021
Accepted: 12 September 2021
Published: 16 September 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Sometimes, it is difficult to distinguish causality from statistical correlation. A prerequisite of causality is the time lag between cause and effect: the cause precedes the effect [1,2]. We consider here causality in a statistical sense, measured by information transfer. Statistical causality direction is inferred from the knowledge of a temporal structure, assuming that the cause has to precede the effect [3]. According to the authors of [4], causal information flow describes the causal structure of a system, whereas information transfer can then be used to describe the emergent computation on that causal structure. For practical reasons, it is convenient to accept that causality can be measured by information transfer, even if the two concepts, are not exactly the same.

Transfer Entropy (TE) is an information transfer measure introduced by Schreiber [5] as a measure used to quantify the statistical coherence between events (usually, time series). Later, TE was considered in the framework of Granger's causality [6,7]. Typically, causality is related to whether interventions on a source have an effect on the target, whereas information transfer is related to whether observations of the source can help predict state transitions on the target. According to Lizier et al. [4], to be considered a correct interpretation of information transfer, TE should only be applied to causal information sources for the given destination. A comparison between the TE and causality indicators can be found, for instance, in [4]. With this in mind, we will use the information transfer (measured by the TE) to establish the presence of and quantify causal relationships.

Massey [8] defined the directivity of information flow through a channel in the form of directed information. In the presence of feedback, this is a more useful quantity than the traditional mutual information. Similarly, the TE measures the information flow from one process to another by quantifying the deviation from the generalized Markov property as a Kullback–Leibler distance. Therefore, the TE can be used to estimate the directional informational interaction between two random variables.

In our case, we quantify the information transfer between the neural layers of feedforward neural architectures. The information between these layers is directed: during the feedforward phase of the backpropagation algorithm, the layers are computed successively. We reduce the weights increment for larger values of TE with the objective of preserving the network configuration if the information is efficiently transferred between layers. The cause (the output from a neural layer) precedes the effect (the input to a subsequent layer). We measure causality by this directional information transfer. The directed informational transfer between two neural layers cannot be measured by a symmetrical statistical measure, such as mutual information, where cause and effect simultaneously coexist.

A variation of the TE is the Transfer Information Energy (TIE), introduced by us in [9,10] as an alternative to the TE. The TE measures of the reduction in uncertainty about one event given another, whereas the TIE measures the increase in certainty about one event given another. The TE and the TIE can be both used as quantitative indicators of information transfer between time series. The TIE can be computed ~20% faster than the TE [9].

Recently, there is a growing interest in applying TE in quantifying the effective connectivity between artificial neurons [11–14]. For instance, the reservoir adaptation method in [15] optimizes the TE at each individual unit, dependent on properties of the information transfer between input and output of the system. Causal relationships within a neural network were defined in [16]. It is a natural question if causal relationships quantified by TE information transfer measure can be inferred from neural networks and included in the learning mechanisms of neural architectures. There are few results reporting applications of TE in training neural networks: Herzog et al. [17,18], Patterson et al. [19], Obst et al. [15], and Moldovan et al. [20].

Herzog et al. [17] computed the feedforward TE between neurons to structure neural feedback connectivity. These feedback connections were used in the training algorithm of a Convolutional Neural Network (CNN) classifier. Herzog et al. averaged (by layer and class) the calculation of TE gathered from directly or indirectly connected neurons, using thresholded activation values. The averaged TEs were then used in the subsequent neuron's activations, within the training algorithm. Only one TE derived value is used for each of the layers. Herzog et al. observed that there is a decreasing feedforward convergence towards higher layers. Furthermore, the TE value is in general lower between nodes at larger layer distances than between neighbors. This is caused by the fact that long-range TE is calculated by conditioning on the intermediate layers.

Herzog et al. continued their research in [18]. Their goal was to define clear guidelines about how to compute the TE based neural feedback connectivity to improve the overall classification performance of feedforward neural network classifiers. Structuring such feedback connection in a deep neural model is very challenging because of the very large number of candidate connections. For AlexNet, Herzog et al. narrowed the TE feedback connections to about 3.5% of all possible feedback connections. Then they used a genetic algorithm to modify their connection strengths and obtain in the end a set of very small weights, similar to many feedback paths in the brain, which amplify already connected feedforward paths only very mildly. According to their experiments in [18], this technique improved the classification performance. In a nutshell, the algorithmic steps in [18] are (i) train the network employing standard backpropagation, (ii) fine-tune the resulted network using feedback TE connections, and (iii) apply a genetic algorithm to generate the best performing network.

Inspired by Herzog et al.'s paper [17], we defined in [20] a novel information-theoretical approach for analyzing the information transfer (measured by TE) between the nodes of feedforward neural networks. The approach employed a different learning mechanism than the one in [17]. The TE was used to establish the presence of relationships and the quantification of these between neurons and the TE values were obtained from neuron output pairs located in consecutive layers. Calculating the TE values required a series of event values that were obtained by thresholding the neurons' outputs with a constant value. We introduced a backpropagation-type training algorithm which used TE feedback connections to improve its performance. Compared with a standard backpropagation training algorithm, the addition of the TE feedback in the training scheme implies a computational overhead needed to compute the TE values in the training stage. However, according to our experiments, adding the TE feedback parameter has three benefits [20]: (a) it accelerates the training process—in general, less epochs are needed; (b) generally achieve a better test set accuracy; and (c) it generates stability during training. The neural models trained in [20] were relatively small. This allowed to use all training samples to compute the TE from time series. When training complex models with large datasets, for computational reasons, this is unpractical. The question (and main motivation for our current work) is how to adapt our technique to such real-world cases.

We extend here the results from [20] and adapt them to a much larger neural architecture—the CNN network. Rather then being a simple generalization, it is a novel approach, as we had to redefine the network training process. The motivation is not only the popularity of CNNs in current deep learning applications, but also the fact that despite the ability of generating human-alike predictions, CNNs still lack a major component: interpretability. CNNs utilize a hierarchy of neural network layers. We consider that the statistical aspects of information transfer between these layers can bring an insight into the feature abstraction process. Our idea is to use TE as a momentum factor in the backward step of backpropagation of error and update the weights in accordance with the uni-directional amount of information transferred between pairs of neurons. We thus leverage the significant informational connection between two units in the learning phase, obtaining a better granularity of the weights' increments.

In contrast to the work in [18], we integrate the TE feedback connections in the training algorithm, and do not use them in a subsequent fine-tuning. The way we select the feedback connections is also different. The similarity between Herzog et al.'s method and our work consists in using TE to compute neural feedback connections. However, the two approaches are very different. Using feedback connections, a general training algorithm like backpropagation adds a computational overhead, not discussed in [17,18]. We may expect that there is a trade-off between execution time and classification accuracy.

Beyond the exploratory aspect of our work, our main insights are twofold. First, we attempt to improve (training time, accuracy) the training algorithm of CNN architectures. Second, we create the premises for further information transfer interpretation within deep learning models.

The rest of the paper is structured as follows. Section 2 introduces the TE definition and notations, whereas Section 3 explains how we compute the TE feedback in a CNN. Section 4 introduces the core of our novel method—the integration of the TE feedback mechanism into the training phase of a CNN. Experimental results are presented in Section 5. Section 6 contains the final remarks and open problems.

2. Transfer Entropy Notations

The recent literature on TE applications is rich and illustrates an increasing interest for the use of this tool in a broad range of applications to measure the directional information flow between time series based on their probability density functions. Applications of TE to date has mainly been concentrated in neuroscience, bioinformatics, artificial life, climate science, finance, and economics [21]. An introduction to TE is offered by Bossomaier et al. [21]. A nonparametric approach of causality considering the conditional entropy was introduced by Baghli [22]. An extensive analysis of causality detection based on information-theoretic approaches in time series analysis is provided in [23].

To introduce the formal definition of TE, let us consider two discrete stationary processes I and J. Relative to time t, k previous data points of process I, l previous data points of process J, and the next point of process I contribute to the definition of TE as follows [5,24]:

$$TE_{J \to I} = \sum_{t=1}^{n-1} p(i_{t+1}, i_t^{(k)}, j_t^{(l)}) \, log \frac{p(i_{t+1}|i_t^{(k)}, j_t^{(l)})}{p(i_{t+1}|i_t^{(k)})}, \tag{1}$$

where $i_t^{(k)}$ and $j_t^{(l)}$ are the k and l dimensional delay vectors of time series I and J, respectively, and i_{t+1} and j_{t+1} are the discrete states at time $t+1$ of I and J, respectively. The generalization of the entropy rate to two processes is a method to obtain a mutual information rate which measures the deviation from independence, and therefore $TE_{J \to I}$ can be obtained from Kullback entropy [5]. TE provides an evaluation of the asymmetric information transfer between two stochastic variables, being a tool which can be used to estimate the unidirectional interaction of pairs of time series.

The precise calculation of the entropy-based measures is an well-known difficult task and the computational effort is still significant when accurate estimation of TE from a dataset is required [25]. One of the most widely used approach is based on the histogram estimation with fixed partitioning, but it is not scalable and is sensible to bins width setting. As TE is derived from the nonparametric entropy estimation, popular methods are widely used for computing the transfer entropy TE [25–27]: kernel density estimation methods, nearest-neighbor, Parzen window density estimation, etc.

3. Computing the TE Feedback in a CNN

CNNs employ a particular form of linear transformation: *convolution*. A convolution operation retains the important and variational features of an input, while flattening the non-variant components. CNN design follows vision processing in living organisms and became very popular starting with the seminal paper of Yann LeCun et al. [28] on handwritten character recognition, where they introduced the LeNet-5 architecture. Since then, research on CNN architectures produced a variety of deep networks, like AlexNet [29], VGG [30], GoogleNet [31], ResNet [32], and more recently EfficientNet [33]. These have many applications which makes them widely used in image recognition and classification, medical imaging, and time series analysis, to name a few.

Despite the ability of generating, especially in image recognition tasks, human-alike predictions, CNNs still lack a major component: interpretability [34]. Neural networks in general are known for their black-box type of behavior, hiding the inner working mechanisms of reasoning. However, reasoning and causal explanations are extremely important for domains like medicine, law, and finance. It is tempting to model statistical cause–effect relationships between CNN neural layers using TE, with the goal to contribute to the interpretability of a CNN. As a first step, even improving the learning algorithm using an information transfer indicator is interesting, as it may lay the ground for future causality explanations.

To quantify inter-neural information transfer in a CNN, we have to quantify the relationship between training the samples and the output values of neurons. The measurable relationship is constructed by selecting subsequent layers and extracting TE values for the pairs of neurons implied. The computed TE values will directly participate in learning mechanism of the network, described in Section 4.

Each TE value is computed by combining two time series, I and J, each obtained by binarization of the activation function of a neuron, with threshold g. Each value in time series I and J is a neuron output computed for an input sample. An ideal binarization threshold should produce only few positive values. The reason is that, in such a case, the obtained TE values tend to have a comparative value with learning rate, and then tend to flatten during CNN training; this gives stability to the learning process. A similar binarization technique was used by Herzog et al. in [17,18].

We compute I and J in Equation (1) for individual pairs of neurons from adjacent layers k and l, l being the next layer after k. Index t is the position of an input sample in the training sequence. Time series I and J are updated online, after processing each input sample. For each considered pair of neurons, the TE value is computed only periodically, after a processing a fixed number of training samples. For all pairs of neurons in layers k and l, we obtain a triangular adjacency matrix of TE values.

It is computationally not feasible to compute the TE values for all possible pairs of neurons. Actually, according to our experiments, not all inter-neural information transfers are relevant for the training process of a network. We observed that the highest impact of using TE in training a CNN is within the last layers. We interpret this as a fine tuning of the classification process, as the classifiable features are available in the final layers of the network. Therefore, we compute the TE values only between the neurons of the last two layers of the CNN. The focus on the last two layers diminishes the computational overhead for calculating the TE values. Our approach is different than the one in Herzog et al. [17,18], where TE interactions between non-adjacent layers are also calculated.

4. TE Feedback Integration in CNN Training

Backpropagation training has two standard phases [35]: feedforward and backward. In the forward phase, for each input sample, in addition to the activations of the neurons for each layer, we also record the time series needed for the TE computation.

The last layer's output is used to calculate the error, which for classification tasks is usually $L = -ln(p_c)$. In the backward phase the weights of error are updated, in reverse order, starting from the last layer. In contrast to the standard backpropagation algorithm, we update the weights with a value resulted from multiplying the current weights by the identity matrix minus the computed TE values. The two phases (feedforward and backward) are alternated until the entire training set is used, and this completes one training epoch. The training consists of several epochs, the training set being randomized at the start of each epoch. In practice, the backpropagation algorithm is used in conjunction with a Mini-batch Gradient Descent (SGD) [36] that updates the weights after a batch of training samples is processed. Mini-batch Gradient Descent is the algorithm of choice for neural networks training. Updating the gradient with the TE addition is synchronized with the batch gradient updates. Without synchronization, the SGD method will diminish the impact of the TE term.

After each batch of training samples goes through forward computation, the backpropagation of errors and new weights computation is performed using Algorithm 1. Different than in the standard backpropagation algorithm [35], we multiply (line 8 of the algorithm) the updated weights by $(\mathbb{I} - (\boldsymbol{te}^{(k)})^\top)$.

Each pair of neurons generates a *te* value; two consequent layers k and l will produce triangular matrix $\mathbf{te}^{(k)}$, used to update the weights for layer k as shown in the Algorithm 1. From line 8 of the algorithm, we conclude that the right member will tamper the weights values, in particular when the cost function has a strong coercive action on the misclassified samples. This accelerates the learning process, as the weights are updated with smaller deltas at the beginning of epochs. Without *te*, the weights receive larger corrections at the beginning of the epoch.

Algorithm 1: Backpropagation using TE for a single step and a single mini-batch. Mini-batches are obtained by equally dividing the training set by a fixed number. This algorithm is repeated for all the available mini-batches and for a number of epochs. Bold items denote matrices and vectors. σ' is the derivative of the activation function σ. The k and l indices are the same as the ones in Equation (1).

```
begin
    foreach layer $k$=1, 2, ..., $l$ do
        $\mathbf{z}^{(k)} = \boldsymbol{w}^{(k)}\boldsymbol{a}^{(k-1)} + \boldsymbol{b}^{(k)}$, where $\mathbf{z}^{(k)}$ is the output of layer $k$ and $\boldsymbol{b}$ is the bias
        $\boldsymbol{a}^{(k)} = \sigma(\mathbf{z}^{(k)})$, where $\boldsymbol{a}^{(k)}$ is the output of the activation function $\sigma$
    end
    foreach layer $k$=$l-1$, ..., 1 do
        $\delta^{(k)} = ((\boldsymbol{w}^{(k+1)})^{\top}\delta^{(k+1)}) \odot \sigma'(\mathbf{z}^{(k)})$, $\delta^{(k)}$ is the layer error, $\odot$ is the
         elementwise product
        $\boldsymbol{w}^{(k)} = (\boldsymbol{w}^{(k)} - \eta\delta^{(k)}\boldsymbol{a}^{(k-1)}) \cdot (\mathbb{I} - (\boldsymbol{te}^{(k)})^{\top})$ where $\mathbb{I}$ is the identity matrix
    end
end
```

5. Experimental Results

All experiments were completed on an AWS Sagemaker ml.p2.xlarge instance using NVIDIA Tesla K80 GPU, with 4 vCPU cores and 61 GB of RAM, engineered on top of PyTorch 1.7.1. The repository is available on GitHub https://github.com/avmoldovan/CNN-TE (accessed on 10 September 2021).

We used the following well-known datasets as benchmarks: CIFAR-10 [37], FashionMNIST [38], STL-10 [39], SVHN [40], and USPS [41] datasets. This selection was determined by the vast amount of literature surrounding it and the number of available implementations and comparisons.

The networks used consist of the following sequential components: convolution and feature engineering, deconvolution and classifier (in this order). Within these, various mechanisms were used to prevent overfitting (e.g., dropout) and obtain normalization.

Our experiments and additions to this architecture involved mainly the classifier part of the network, but we have also ran experiments on the convolutional layers.

We applied the TE on the last (fully connected) two layers—the pre-softmax and softmax layers—with different binarization thresholds determined experimentally (see Table 1). The TE term is applied on the weights of the k-th layer (see the red arrows in Figure 1).

The softmax layer transforms the outputs of a linear layer into probabilities. The maximum probability corresponds to the predicted class. For all outcomes with probability above the threshold, the J time-series are positive.

Figure 2 depicts the architecture of the network used for the USPS dataset.

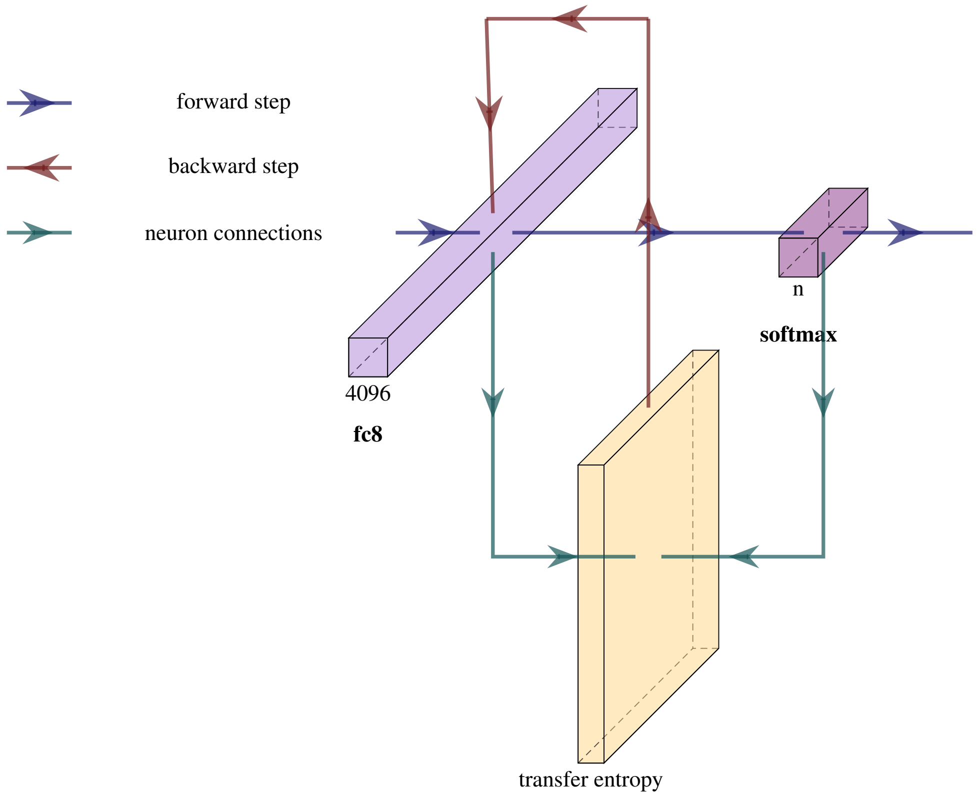

Figure 1. During the feedforward step, we compute time series I and J, and the **te** matrix, as shown by the green arrows. When the backward step propagates the errors, we then use the **te** matrix in the weight updates as shown in the Algorithm 1.

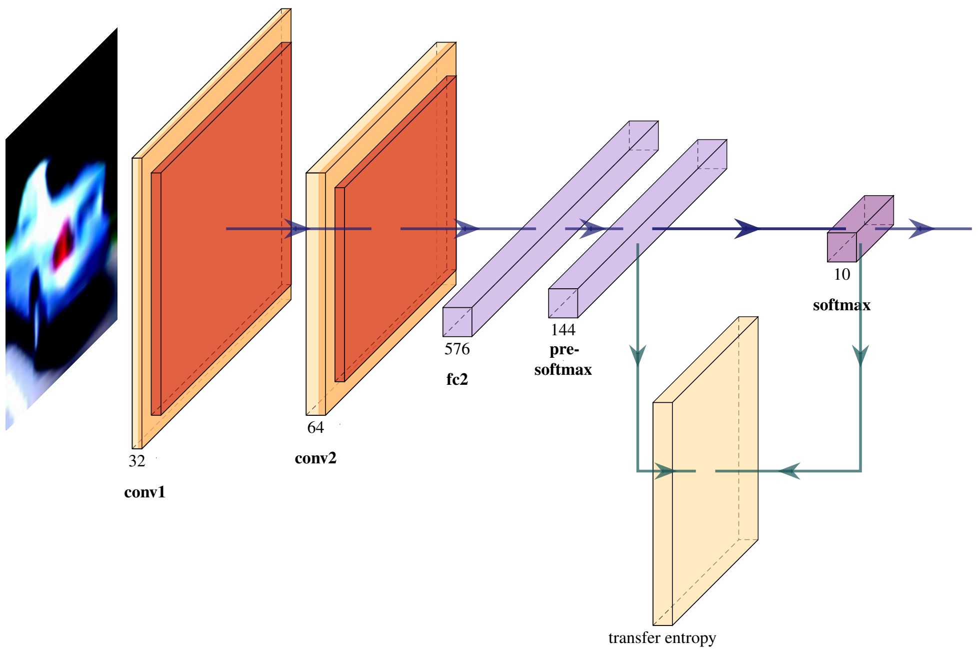

Figure 2. Illustration of the feedforward phase for the USPS dataset. The green arrows indicate the layers outputs that are used to compute the TE (Plotted using https://github.com/HarisIqbal88/PlotNeuralNet (accessed on 10 September 2021)).

During training, at the beginning of each epoch, we noticed an increased instability, visible through the high variation and values of the gradients, as seen in Figure 3. These observation apply for all datasets and networks, with or without the TE added. The TE values also exhibit instability and have larger values at the beginning of each epoch. However, the TE values show smaller values during the first epochs due to the selected threshold value that matches larger weights values from subsequent epochs. During each epoch, and also during the whole training process, the slope of the gradients gradually decreases and the TE variation also decreases.

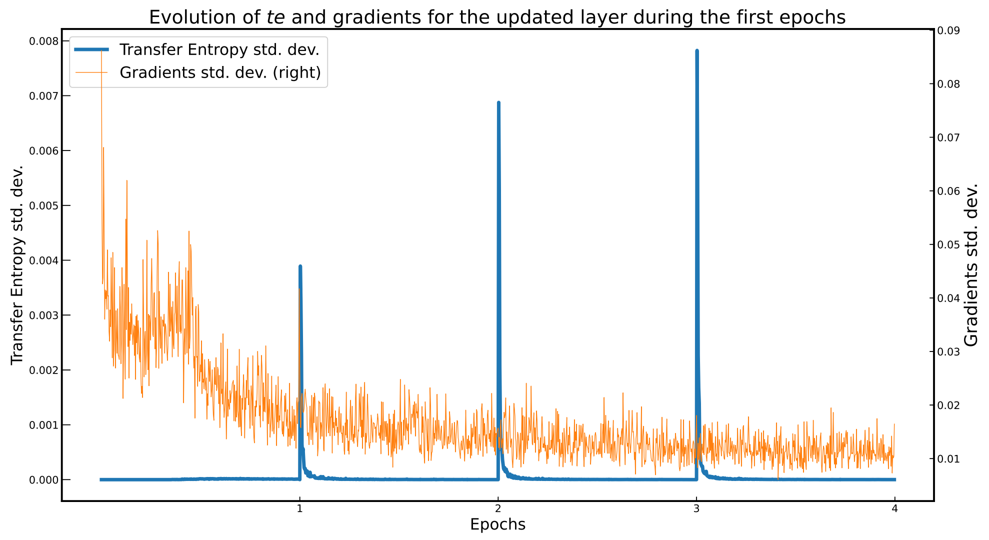

Figure 3. Evolution of the **te** standard deviation values on the first 4 epochs for the SVHN+TE dataset, for the pre-softmax layer. Each data point in the plot represents a batch. The rest of the TE values have a similar shape and decrease slowly during training. We observe the spikes of the TE values at the beginning of each epoch due to the training set randomization. During the first epoch the TE values are not calculated for the first batches in order to prevent anomalous values, thus its value is close to 0.

To validate the TE impact, we set a target accuracy to be reached by both implementations with/without TE. We observed the implementation that reaches the target accuracy w.r.t. the number of epochs needed, as well as the average time per epoch. These results show which of the two implementation requires less epochs to reach a target accuracy on the test set. For a fair comparison, we used the same hyperparameters for the CNNs with/without TE implementations (see Table 1). The results are summarized in Tables 2–6.

Table 1. Parameters.

	CIFAR-10+TE	**FashionMNIST+TE**	**STL-10+TE**	**SVHN+TE**	**USPS+TE**
learning rate (η)	0.01	0.01	0.01	0.01	0.01
momentum	0.9	0.9	0.9	0.9	0.9
dropout	0	0.25	0.	0.3	0.25
threshold rate 1 (g_1)	2.0	2.0	2.0	2.0	5.0
threshold rate 2 (g_2)	0.99	0.99	0.99	0.99	0.99
te window length	100	100	4000	200	90
batch size	500	100	200	200	60

Table 2. Results for CIFAR-10 [37], with/without TE. The increased training time results from the large size of the last linear layer.

	CIFAR-10+TE	**CIFAR-10**
Target **98%** accuracy in epoch	**5**	6
Top 1 accuracy at epoch 5	**98.02%**	97.58%
Average epoch duration	2110 s	**81 s**
Total training duration	10,550 s	**492 s**

Table 3. Results for Fashion-MNIST [38] dataset, with/without TE.

	FashionMNIST+TE	**FashionMNIST**
Target **97%** accuracy in epoch	**23**	28
Top 1 accuracy at epoch 23	**97.0%**	97.02%
Average epoch duration	71 s	**41 s**
Total training duration	1720 s	**1162 s**

Table 4. Results for STL-10 [39] dataset, with/without TE.

	STL-10+TE	**STL-10**
Target **98%** accuracy in epoch	**5**	7
Top 1 accuracy at epoch 5	**98.33%**	78.63%
Average epoch duration	28 s	**7 s**
Total training duration	128 s	**53 s**

Table 5. Results for SVHN [40] dataset, with/without TE.

	SVHN+TE	**SVHN**
Target **94%** accuracy in epoch	**9**	11
Top 1 accuracy at epoch 9	**94.05%**	91.67%
Average epoch duration	512 s	**491** s
Total training duration	**4587** s	5369 s

Table 6. Results for USPS [41] dataset, with/without TE.

	USPS+TE	**USPS**
Target **99%** accuracy in epoch	3	3
Top 1 accuracy at epoch 3	**99.32**%	99.05%
Average epoch duration	376 s	**33 s**
Total training duration	1138 s	**102** s

As observed in [20], using time series constructed from the full length of the epoch results in smoother TE values. Computing the TE for large time series (e.g., $>10^6$) is computationally impractical. We also observed the necessary length of the time series in that produces observable and positive outcomes. Therefore, we limited this length to u (determined experimentally), using a sliding window technique. To obtain smoother TE values, we slid the window with every batch of training samples, instead of dividing the training set into u-sized windows. Figure 4 illustrates how the time series were computed.

Using this approach, we analyzed the impact of the window length on the accuracy of the classifier. After q training samples, we computed the TE. The u windows overlap partially, as shown in Figure 4. According to our experiments, limiting the length of the time series does not have a significant impact on the performance of the trained classifier. Furthermore, as we found that a u value five times the batch size is a good trade-off between accuracy and computational overhead.

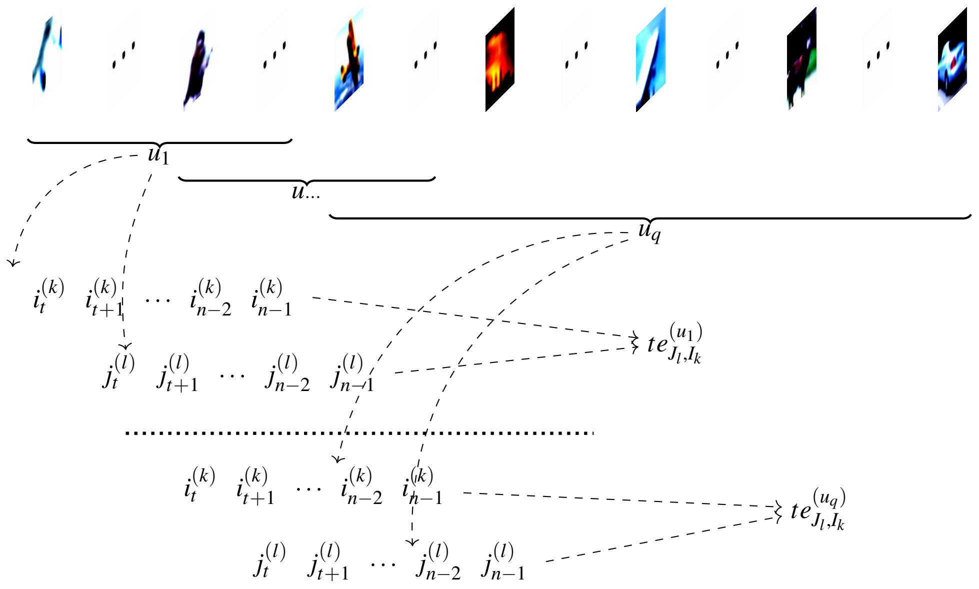
u_1
$u_{\dots}$
u_q
$i_t^{(k)}$ $i_{t+1}^{(k)}$ $\cdots$ $i_{n-2}^{(k)}$ $i_{n-1}^{(k)}$
$j_t^{(l)}$ $j_{t+1}^{(l)}$ $\cdots$ $j_{n-2}^{(l)}$ $j_{n-1}^{(l)}$
$te_{J_l,I_k}^{(u_1)}$
$i_t^{(k)}$ $i_{t+1}^{(k)}$ $\cdots$ $i_{n-2}^{(k)}$ $i_{n-1}^{(k)}$
$j_t^{(l)}$ $j_{t+1}^{(l)}$ $\cdots$ $j_{n-2}^{(l)}$ $j_{n-1}^{(l)}$
$te_{J_l,I_k}^{(u_q)}$

Figure 4. Illustration of how time series I and J are produced for a pair of neurons from layers k and l, for multiple windows of events $u_1 \dots u_q$.

Convolutional layers are a major building blocks used in CNNs [42]. A convolution is performed on the input data with the use of a kernel to produce a feature map. Applying the TE to measure the inter-neural information transfers between the input data and the resulted feature maps is interesting to be considered. We can compute the median of the activations of the neurons within each convolutional kernel from layer *conv1* (see Figure 2) and pair it with the outputs of subsequent layer *conv2*. The obtained *te* values can be used in the CNN learning process. In our experiments, under this setup, the learning process diverged. In the best run, the top 1 accuracy hardly reached a considerable value. In addition, this approach has a considerable computational overhead, especially if we consider several convolutional layers. This justifies our focus on the last two fully connected layers only.

It is also interesting to evaluate the impact the length s of the time series. Experimentally, we observed that when s is a multiple of the batch size b, the accuracy maintains a favorable trend. In this scenario, the time series are constructed as illustrated in Figure 4. The best results were obtained when the length of the series are extended to the full epoch.

In another set of experiments, we tried to minimize the number of considered neuron pairs from the last two layers, with a minimum impact on the achieved accuracy. In other words, we tried to obtain an optimal performance–computational overhead trade-off. We found that the accuracy improves significantly even when using only 10% of the randomly selected neurons. The neurons were selected randomly for an entire epoch or by a TE window. The two strategies yielded similar results. This is somehow similar to dropout, as only some of the connections are updated using the TE feedback. The top performance is achieved when all neurons are selected.

According to our experiments, using the TE feedback loops for additional layer pairs improves the performance. However, this increases exponentially the number of TE computations needed. For the USPS network (see Appendix A.5), a very simple dataset, computing the TE for the last two linear layers adds an overhead of 7 min of training time per epoch. Computing the TE only for the pre-softmax and softmax layers training takes 6 min per epoch. Computing the TE for the convolutional layers for the USPS network implies an increased computational overhead. For all possible pairs of kernels between the convolutional layers, we measured an extra three days of training per epoch. Performance cost increases almost linearly with the number of performed TE calculations. The exact number of TE computations for a pair of layers is a product of the layer sizes and the number of batches. Therefore, it is computationally not practical to compute the TE for all layers.

We also conducted other experiments, applying the TE correction on an identical network, pre-trained without the TE mechanism. We continued training, freezing all layers (except the pre-softmax and softmax layers), and applying the TE correction. The results were not consistent through multiple executions, since the TE training on top of the non-TE training changes the convergence logic and creates instability.

6. Conclusions and Open Problems

TE can be used to measure how significantly neurons interact [11,12]. In our study, we add specific TE feedback connections between neurons to improve performance. Our results confirm what we obtained in our previous study on a simple feedforward neural architecture [20]. Adding the TE feedback parameter accelerates the training process, as fewer epochs are needed. On the flip side, it adds computational overhead to each epoch. The optimal balance between these two conflicting factors is application dependent.

According to our results, in a CNN classifier it is efficient to consider only the inter-neural information transfer of the neuron pairs between the last two fully connected layers. The information transfer within these layers has the most significant impact on the learning process, since they are the closest to the high-level classification decision. Many of the inter-neural information transfer connections appear to be redundant, and this allows us to use only a fraction of them. These observations are very interesting and may be further discussed in from a neuroscientific perspective (e.g., the vertebrate brain [43,44]).

Generally, to optimize the generalization of a learning algorithm, we try to minimize the number of its parameters. As adding the TE in the learning mechanism generates new hyper-parameters, connected to the integration of the TE in the learning algorithm, the question is if this does not conduct to overfitting and a weaker generalization performance. In our experiments, the TE acted as a smoothing factor, becoming active only periodically, not after processing each input sample. Therefore, we can consider the TE is in our model a slowly changing meta-parameter. This can be related to the hierarchy of quickly-changing vs. slowly-changing parameters in learning neural causal models [45]. We observed that the TE feedback generates stability during training, this being compliant with the results presented in [20].

According to the authors of [18], it is tempting to speculate that a similar principle—an evaluation of the relevance of the different feedforward pathways—might have been a phylo- or ontogenetic driving force for the design of different feedback structures in real neural systems.

Author Contributions: A.M., A.C. and R.A. equally contributed to the published work. All authors have read and agreed to the published version of the manuscript.

Funding: The costs to publish in open access were covered by Siemens SRL.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

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

Abbreviations

The following abbreviations are used in this manuscript:

te	transfer entropy value
TE	Transfer Entropy
g	the binarization threshold
u	window of time series used to calculate TE
W	weights matrix
C	loss function
CNN	Convolutional Neural Network
SGD	Stochastic Gradient Descent
CNN+TE	CNN + Transfer Entropy—our proposed method
MLP	Multi-layer perceptron

Appendix A. CNN Architectures

Appendix A.1. FashionMNIST

The architecture used for the FashionMNIST [38] dataset is https://www.kaggle.com/pankajj/fashion-mnist-with-pytorch-93-accuracy (accessed on 10 September 2021):

```
Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Linear(in_features=2304, out_features=600, bias=True)
Dropout(p=0.25, inplace=False)
Linear(in_features=600, out_features=120, bias=True)
Linear(in_features=120, out_features=10, bias=True)
Softmax(dim=1)
```

Appendix A.2. CIFAR-10

For CIFAR-10 [37] dataset we used the following layout https://github.com/aaron-xichen/pytorch-playground (accessed on 10 September 2021):

```
Conv2d(3, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(128, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(128, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(256, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(256, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(512, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(512, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1))
BatchNorm2d(1024, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Linear(in_features=1024, out_features=10, bias=True)
Softmax(dim=1)
```

Appendix A.3. STL-10

For the STL-10 dataset [39] we used the following network architecture https://github.com/aaron-xichen/pytorch-playground (accessed on 10 September 2021):

```
Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(32, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(64, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(128, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(128, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
BatchNorm2d(256, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1))
BatchNorm2d(256, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Linear(in_features=256, out_features=10, bias=True)
Softmax(dim=1)
```

Appendix A.4. SVHN

For the SVHN dataset [40] we used the following network architecture https://github.com/aaron-xichen/pytorch-playground (accessed on 10 September 2021):

```
Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(32, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
Dropout(p=0.3, inplace=False)
Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(32, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
Dropout(p=0.3, inplace=False)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(64, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
Dropout(p=0.3, inplace=False)
Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(64, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
Dropout(p=0.3, inplace=False)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(128, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
Dropout(p=0.3, inplace=False)
Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(128, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
Dropout(p=0.3, inplace=False)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
BatchNorm2d(256, eps=1e-05, momentum=0.1)
ReLU(inplace=True)
Dropout(p=0.3, inplace=False)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Linear(in_features=256, out_features=10, bias=True)
Softmax(dim=1)
```

Appendix A.5. USPS

For the USPS dataset [41] we used the following network architecture https://github.com/aaron-xichen/pytorch-playground (accessed on 10 September 2021):

```
Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
ReLU(inplace=True)
MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
Linear(in_features=576, out_features=144, bias=True)
```

```
Dropout(p=0.25, inplace=False)
Linear(in_features=144, out_features=10, bias=True)
Softmax(dim=1)
```

References

1. Shadish, W.; Cook, T.; Campbell, D. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*; Houghton Mifflin: Boston, MA, USA, 2001.
2. Marwala, T. *Causality, Correlation and Artificial Intelligence for Rational Decision Making*; World Scientific: Singapore, 2015. [CrossRef]
3. Zaremba, A.; Aste, T. Measures of Causality in Complex Datasets with Application to Financial Data. *Entropy* **2014**, *16*, 2309–2349. [CrossRef]
4. Lizier, J.T.; Prokopenko, M. Differentiating information transfer and causal effect. *Eur. Phys. J. B* **2010**, *73*, 605–615. [CrossRef]
5. Schreiber, T. Measuring Information Transfer. *Phys. Rev. Lett.* **2000**, *85*, 461–464. [CrossRef] [PubMed]
6. Barnett, L.; Barrett, A.B.; Seth, A.K. Granger Causality and Transfer Entropy Are Equivalent for Gaussian Variables. *Phys. Rev. Lett.* **2009**, *103*, 238701. [CrossRef]
7. Hlaváčková-Schindler, K. Equivalence of Granger Causality and Transfer Entropy: A Generalization. *Appl. Math. Sci.* **2011**, *5*, 3637–3648.
8. Massey, J.L. Causality, feedback and directed information. In Proceedings of the 1990 International Symposium on Information Theory and its applications and Its Applications, Honolulu, HI, USA, 27–30 November 1990; pp. 303–305.
9. Caţaron, A.; Andonie, R. Transfer Information Energy: A Quantitative Causality Indicator Between Time Series. In Proceedings of the Artificial Neural Networks and Machine Learning—ICANN 2017—26th International Conference on Artificial Neural Networks, Alghero, Italy, 11–14 September 2017; pp. 512–519. [CrossRef]
10. Caţaron, A.; Andonie, R. Transfer Information Energy: A Quantitative Indicator of Information Transfer between Time Series. *Entropy* **2018**, *20*, 323. [CrossRef]
11. Lizier, J.T.; Heinzle, J.; Horstmann, A.; Haynes, J.D.; Prokopenko, M. Multivariate information-theoretic measures reveal directed information structure and task relevant changes in fMRI connectivity. *J. Comput. Neurosci.* **2011**, *30*, 85–107. [CrossRef]
12. Vicente, R.; Wibral, M.; Lindner, M.; Pipa, G. Transfer entropy—A model-free measure of effective connectivity for the neurosciences. *J. Comput. Neurosci.* **2011**, *30*, 45–67. [CrossRef] [PubMed]
13. Shimono, M.; Beggs, J.M. Functional Clusters, Hubs, and Communities in the Cortical Microconnectome. *Cereb. Cortex* **2015**, *25*, 3743–3757. [CrossRef]
14. Fang, H.; Wang, V.; Yamaguchi, M. Dissecting Deep Learning Networks—Visualizing Mutual Information. *Entropy* **2018**, *20*, 823. [CrossRef]
15. Obst, O.; Boedecker, J.; Asada, M. Improving Recurrent Neural Network Performance Using Transfer Entropy. In *Proceedings of the 17th International Conference on Neural Information Processing: Models and Applications—Volume Part II*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 193–200.
16. Féraud, R.; Clérot, F. A methodology to explain neural network classification. *Neural Netw.* **2002**, *15*, 237–246. [CrossRef]
17. Herzog, S.; Tetzlaff, C.; Wörgötter, F. Transfer entropy-based feedback improves performance in artificial neural networks. *arXiv* **2017**, arXiv:1706.04265
18. Herzog, S.; Tetzlaff, C.; Wörgötter, F. Evolving artificial neural networks with feedback. *Neural Netw.* **2020**, *123*, 153–162. [CrossRef]
19. Patterson, J.; Gibson, A. *Deep Learning: A Practitioner's Approach*, 1st ed.; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2017.
20. Moldovan, A.; Caţaron, A.; Andonie, R. Learning in Feedforward Neural Networks Accelerated by Transfer Entropy. *Entropy* **2020**, *22*, 102. [CrossRef]
21. Bossomaier, T.; Barnett, L.; Harré, M.; Lizier, J.T. *An Introduction to Transfer Entropy. Information Flow in Complex Systems*; Springer: Berlin/Heidelberg, Germany, 2016.
22. Baghli, M. A model-free characterization of causality. *Econ. Lett.* **2006**, *91*, 380–388. [CrossRef]
23. Hlaváčková-Schindler, K.; Paluš, M.; Vejmelka, M.; Bhattacharya, J. Causality detection based on information-theoretic approaches in time series analysis. *Phys. Rep.* **2007**, *441*, 1–46. [CrossRef]
24. Kaiser, A.; Schreiber, T. Information transfer in continuous processes. *Phys. D Nonlinear Phenom.* **2002**, *166*, 43–62. [CrossRef]
25. Gencaga, D.; Knuth, K.H.; Rossow, W.B. A Recipe for the Estimation of Information Flow in a Dynamical System. *Entropy* **2015**, *17*, 438–470. [CrossRef]
26. Hlaváčková-Schindler, K. Causality in Time Series: Its Detection and Quantification by Means of Information Theory. In *Information Theory and Statistical Learning*; Emmert-Streib, F., Dehmer, M., Eds.; Springer: Boston, MA, USA, 2009; pp. 183–207. [CrossRef]
27. Zhu, J.; Bellanger, J.J.; Shu, H.; Le Bouquin Jeannès, R. Contribution to Transfer Entropy Estimation via the k-Nearest-Neighbors Approach. *Entropy* **2015**, *17*, 4173–4201. [CrossRef]
28. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]

29. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*; Curran Associates, Inc.: Red Hook, NY, USA, 2012.
30. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2014**, arXiv:1409.1556.
31. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.E.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going Deeper with Convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp 1–9.
32. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016.
33. Tan, M.; Le, Q.V. EfficientNet: R ethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019.
34. Muşat, B.; Andonie, R. Semiotic Aggregation in Deep Learning. *Entropy* **2020**, *22*, 1365. [CrossRef] [PubMed]
35. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*; MIT Press: Cambridge, MA, USA, 1986; Volume 1, pp. 318–362.
36. Shalev-Shwartz, S.; Singer, Y.; Srebro, N.; Cotter, A. Pegasos: Primal estimated sub-gradient solver for SVM. *Math. Program.* **2011**, *127*, 3–30. [CrossRef]
37. Krizhevsky, A. *Learning Multiple Layers of Features from Tiny Images*; Technical Report; University of Toronto: Toronto, ON, Canada, 2009.
38. Xiao, H.; Rasul, K.; Vollgraf, R. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv* **2017**, arXiv:1708.07747.
39. Coates, A.; Ng, A.; Lee, H. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. *J. Mach. Learn. Res. Proc. Track* **2011**, *15*, 215–223.
40. Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; Ng, A. Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS* **2011**, 1–9.
41. Hull, J.J. A database for handwritten text recognition research. *IEEE Trans. Pattern Anal. Mach. Intell.* **1994**, *16*, 550–554. [CrossRef]
42. Lecun, Y.; Haffner, P.; Bottou, L.; Bengio, Y. Object Recognition with Gradient-Based Learning. In *Contour and Grouping in Computer Vision*; Springer: Berlin/Heidelberg, Germany, 1999.
43. Gilbert, C.D.; Li, W. Top-down influences on visual processing. *Nat. Rev. Neurosci.* **2013**, *14*, 350–363. [CrossRef] [PubMed]
44. Spillmann, L.; Dresp-Langley, B.; Tseng, C.H. Beyond the classical receptive field: The effect of contextual stimuli. *J. Vis.* **2015**, *15*, 7. [CrossRef] [PubMed]
45. Ke, N.R.; Bilaniuk, O.; Goyal, A.; Bauer, S.; Larochelle, H.; Pal, C.; Bengio, Y. Learning Neural Causal Models from Unknown Interventions. *arXiv* **2019**, arXiv:1910.01075.