


Tutorial

# Hands-On Fundamentals of 1D Convolutional Neural Networks—A Tutorial for Beginner Users

Ilaria Cacciari \*  and Anedio Ranfagni

Istituto di Fisica Applicata “Nello Carrara”, Consiglio Nazionale delle Ricerche, Via Madonna del Piano 10, 50019 Sesto Fiorentino, Italy

\* Correspondence: i.cacciari@ifac.cnr.it

**Abstract:** In recent years, deep learning (DL) has garnered significant attention for its successful applications across various domains in solving complex problems. This interest has spurred the development of numerous neural network architectures, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Generative Adversarial Networks (GANs), and the more recently introduced Transformers. The choice of architecture depends on the data characteristics and the specific task at hand. In the 1D domain, one-dimensional CNNs (1D CNNs) are widely used, particularly for tasks involving the classification and recognition of 1D signals. While there are many applications of 1D CNNs in the literature, the technical details of their training are often not thoroughly explained, posing challenges for those developing new libraries in languages other than those supported by available open-source solutions. This paper offers a comprehensive, step-by-step tutorial on deriving feedforward and backpropagation equations for 1D CNNs, applicable to both regression and classification tasks. By linking neural networks with linear algebra, statistics, and optimization, this tutorial aims to clarify concepts related to 1D CNNs, making it a valuable resource for those interested in developing new libraries beyond existing ones.

**Keywords:** convolutional neural network; linear algebra; statistics; tutorial



**Citation:** Cacciari, I.; Ranfagni, A. Hands-On Fundamentals of 1D Convolutional Neural Networks—A Tutorial for Beginner Users. *Appl. Sci.* **2024**, *14*, 8500. <https://doi.org/10.3390/app14188500>

Academic Editor: Douglas O’Shaughnessy

Received: 20 August 2024

Revised: 12 September 2024

Accepted: 19 September 2024

Published: 20 September 2024



**Copyright:** © 2024 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

Recently, there has been a surge of interest in deep learning (DL), with numerous review papers on the topic published in recent years [1–14]. DL architectures, such as Deep Neural Networks (DNNs), Deep Belief Networks (DBNs), Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Transformers, have found compelling applications across various fields. These include computer vision, speech recognition [15–21], natural language processing [22–29], bioinformatics [30–37], medical image analysis [38–47], climate science [48–58], and material inspection [59–67].

Focusing specifically on one-dimensional problems, CNN-based architectures are still the most widely used in the literature compared to other, more recent, and potentially more effective architectures for certain problems. This observation is based on research conducted using prominent databases and search engines (e.g., Scopus, IEEExplore, ScienceDirect, SpringerLink, Web of Science, ACM Digital Library, PubMed). Relevant results have emerged regarding the frequency with which a specific deep learning architecture, associated with one-dimensional problems, appears in paper titles published between 2018 and 2023. Although this research is not exhaustive, it highlights that discussions on 1D CNNs remain highly active in the literature.

However, within the realm of CNNs applications, it is important to note that those focusing on one-dimensional data are less frequent compared to those dealing with multidimensional data (2D and 3D). This discrepancy in research activity can be observed through a simple Google search. For example, a search for terms like “3D CNN”, “2D CNN”, or “1D CNN” reveals a significant drop in the number of results as the dimensionality of the data

fed into the network decreases. Although searches include web pages and blog posts, which are generally not peer-reviewed, a similar trend is evident in the peer-reviewed literature as well. The relatively limited number of published papers on 1D CNNs, compared to those on 2D and 3D, suggests an under-explored area that could stimulate further research [68]. One possible outcome of this could be an increase in review papers specifically focusing on 1D CNNs, which are currently quite limited [69–71].

1D CNNs offer promising solutions across various sectors. They are particularly well-suited for real-time applications [72–75] due to their low computational requirements. In fact, continuous online monitoring generates a large quantity of data that needs to be processed and analyzed rapidly and accurately to efficiently detect specific patterns, assets or even diseases. This is the case of Raman spectra detection, in which the use of 1D CNNs has proven effective at identifying in real-time one pure unknown Raman instance from thousands of classes with higher accuracy over traditional multivariate analyses [72]. In the context of real-time applications for 5G communications, the increased bandwidth has posed a significant challenge in linearization of power amplifier over a wide bandwidth, which has been faced implementing 1D CNNs that have proven to be highly effective for real-time digital pre-distortion operation, surpassing traditional methods like Volterra series [73].

The use of 1D CNNs has also been proposed as a computationally efficient solution for fast and accurate diagnostics in the power industry, particularly for monitoring electromagnetic interference. This approach has enabled highly accurate automatic condition monitoring, significantly aiding in the identification of insulation and mechanical faults in high-voltage electrical power assets [74]. Another interesting example the real-time processing capability with significantly lower delay achieved by using 1D CNNs has been presented to classify cardiac acoustic signals for heart anomaly detection. This method proved to be robust and accurate also in reducing the false alarms [75].

The ability of 1D CNNs to learn models from 1D signals such as acoustic signals has also been well demonstrated. Specifically, 1D CNNs are increasingly being used to process raw audio waveforms for environment sound [76], music genre [77], and in general audio [78] classification, along with speech acoustic modeling [79], garnering growing attention in these areas.

Given that 1D CNNs can process any 1D signal, they are also applicable to physiological signals such as electroencephalograms (EEG) and electrocardiograms (ECG). Therefore, fields traditionally dominated by 2D CNNs, such as medicine and biology, could benefit from the application of 1D CNNs. A significant portion of the literature on classification problems involves time series signals as input data, with outputs typically predicting labels related to brain states for tasks such as identifying epileptic episodes [80,81], establishing communication paths between brain processes and external devices [82], and classifying sleep phases [83]. Additionally, there are notable examples of classification of various cardiac disease from ECGs [84–87], and or other conditions, including Parkinson's disease [88], COVID-19 [89], and diabetes [90].

Moreover, 1D CNNs have likewise become popular for improving intrusion detection in cyber-physical systems [91–102], and, within industrial contexts, particularly for fault and anomaly detection in complex production environments [103,104].

Additionally, 1D CNNs have proven valuable for cognitive tasks, such as classifying emotional states, including speech emotion recognition. They have been shown to be effective for speaker-independent types of emotion recognition [105], offering increased accuracy while maintaining reasonable computational costs [106,107].

Noteworthy applications of 1D CNNs extend to the financial sector, where they can be used to identify line chart patterns from financial time series [108]. In spectroscopy, 1D CNNs have demonstrated intriguing applications, such as predicting Dry Matter Content (DMC) from infrared absorbance spectra of fruits—a crucial indicator of fruit maturity that impacts harvest timing—[109], and classifying corn seed viability [110].

Moreover, in the biomedical field, 1D CNNs have been effectively used to extract

lifetime parameters from fluorescence lifetime imaging microscopy signals [111]. In agricultural environmental monitoring, 1D CNNs have been applied to classify soil texture using hyper-spectral data [112–114] and to predict various soil properties from visible and infrared spectra [115]. In geology, 1D CNNs have demonstrated their ability to link multi-variate geochemical data with known mineral deposit locations, significantly enhancing the precision of exploration areas [116]. They also offer rapid seismic predictions, potentially addressing the limitations of traditional physics-informed systems [117], and can be used to classify seismic signals from different tectonic settings [118].

While this literature review is not exhaustive, it highlights that 1D CNNs continue to be widely utilized within the community. This raises the question of whether their prevalent use, despite the availability of potentially more efficient architectures, is due to a deep understanding of their fundamental mechanisms by researchers or the convenience of using readily available open-source software packages. Although many applications have already been developed and more are on the horizon, there appears to be a gap. Among common themes and recurring issues observed in the literature, we find that many published papers, even if innovative from an application standpoint and highly cited, often lack detailed descriptions of fundamental theoretical aspects. While the use of open-source packages is frequently mentioned, very little attention is generally paid to theoretical aspects such as which definition of convolution has been considered, which loss or activation functions have been used, and whether some kind of padding has been introduced. In those papers where a theoretical section exists, many technical aspects are only briefly outlined. In other words, the success of 1D CNNs partly stems from gaps in knowledge that do not impede progress as long as pre-existing software solutions are utilized. This is evidenced by the ongoing activity within the community on the topic.

However, the situation becomes more challenging if one needs to develop custom neural network libraries in languages other than those supported by available open-source solutions. Implementing a 1D CNN-based solution in new libraries written in different languages presents several potential challenges. A deep understanding of fundamental concepts such as convolutions, activation functions, pooling operations, and backpropagation is essential.

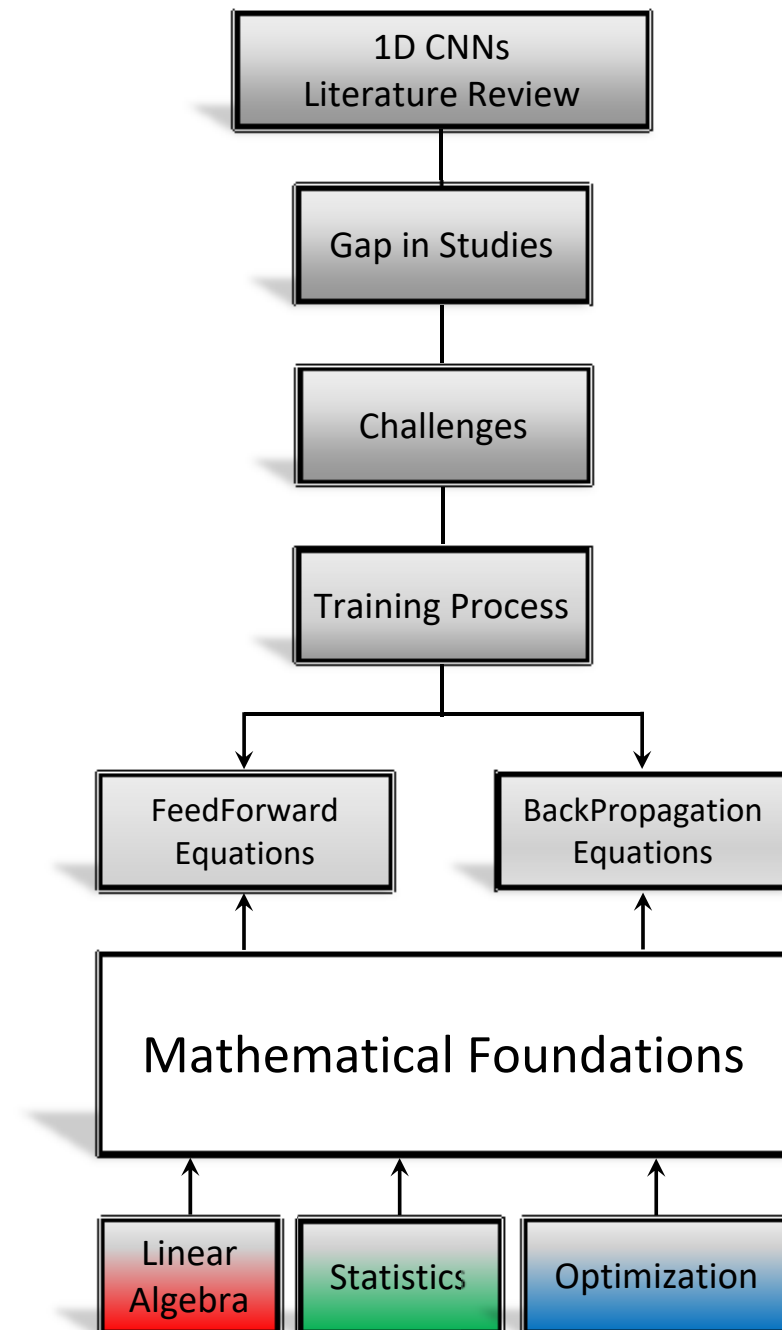
However, this knowledge alone is not enough to tackle the challenges that such a development requires. It is necessary to ensure that the new libraries are accompanied by thorough documentation that explains the theoretical underpinnings of 1D CNNs in detail. Additionally, educational resources, including tutorials, code examples, and step-by-step guides, are necessary to help users grasp the implementation process. Finally, community support is also vital for the successful adoption of these tools, including not only scholarly publications but also forums or user groups where developers can exchange ideas and seek assistance.

On a practical level, once the theoretical foundations are established, attention must be given to various other factors such as performance optimization, compatibility with existing systems, cross-platform support, and fostering user adoption to build a supportive community. Moreover, developing new libraries for 1D CNNs involves potential risks and challenges, including overfitting, generalization issues, data requirements, and computational complexity. While these concerns are important for the developer community, they are beyond the scope of this tutorial.

Although the general theory of CNNs is well-covered in books and papers (especially for 2D CNNs), in the few that specifically refer to 1D CNNs [69,119–122], there are still theoretical elements that are not sufficiently outlined but that deserve further explanation. This tutorial aims to bridge what is still missing, providing a step-by-step guide that includes all the necessary mathematical expressions for using 1D CNNs. It starts with a clear definition of convolution, which can sometimes be confusing, and progresses through the feedforward and backpropagation equations. The scope is to present these expressions explicitly, helping readers to become more familiar with the numerous neural network libraries available and, over time, aiding in the development of new libraries in languages

other than those currently available.

The tutorial covers each aspect of feedforward and backpropagation in 1D CNNs. It particularly addresses limitations noted in our literature review, such as the limited discussion of activation functions (typically only hyperbolic tangent or sigmoid functions are covered) and loss functions (which are often chosen for regression rather than classification tasks). To tackle these issues, we draw on concepts from linear algebra, statistics, and optimization, as schematically depicted in Figure 1.



**Figure 1.** Our literature review on 1D CNNs has revealed gaps in studies, posing a challenge in fully addressing the theoretical elements required to derive all the equations involved in the training process (i.e., feedforward and backpropagation). The mathematical foundations we refer to include linear algebra, statistics, and optimization.

The tutorial is structured as follows. In Section 2, discrete convolution is defined with an emphasis on its distinction from cross-correlation. In Section 3, the network architecture is described in terms of building blocks. The training process is discussed in Section 4 for the feedforward propagation, and in Section 5 for the backpropagation. Section 6 explores other additional activation functions, such as hyperbolic tangent (tanh), Rectified Linear Unit (ReLU) and logarithmic hyperbolic cosine (log cosh), as well as other loss functions for both regression and classification tasks. In Section 7, the conclusions are presented, and Appendix A provides a review of the derivatives used in Section 5 (backpropagation section).

### 2. Convolution

In general, discrete convolution involves sliding a filter ( $K$ ) over an input signal ( $I$ ). The simplest case is when the filter slides over each position of the input.

Assuming that the filter iterates one element at a time (i.e., stride = 1), this operation can be performed according to the following definitions:

$$(I * K)(i) = \sum_{u=1}^s I(i + u - 1)K(u) \tag{1}$$

$$(I * K)(i) = \sum_{u=1}^s I(i - u - 1)K(u) \tag{2}$$

where  $I$  is the input of length  $n$  and  $K$  the filter (kernel) length  $s$ .

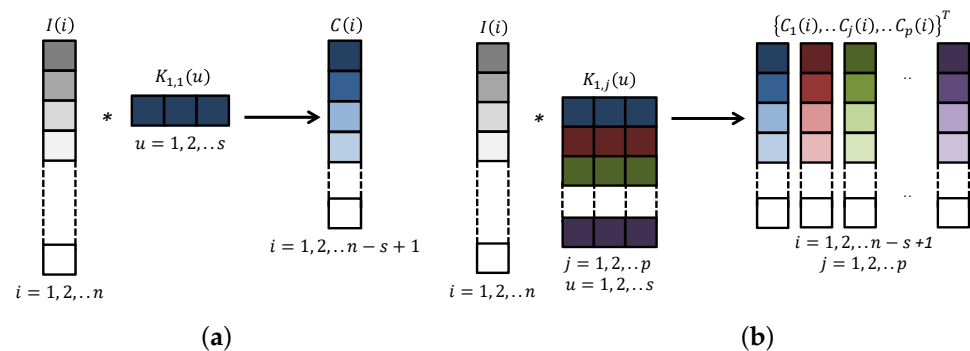
Equation (1) is known as non-causal convolution and, in the context of digital signal processing, as cross-correlation; (2) is known as causal convolution. According to definition (1), cross-correlation means sliding a filter across an input signal, while convolution (2) means sliding a flipped filter across the input signal [123].

In modern CNN packages (such as PyTorch [124]), the operation commonly referred to as convolution is actually cross-correlation (1).

However, this misnomer does not impact the performance of the network since the network has the ability to adapt accordingly. This is true as long as ready-made packages are used to implement a CNN. If not, to avoid confusion, it is worth specifying which convolution definition is implemented. Here, 1D CNNs are explored using the convolution definition (1), and in the following, we refer to definition (1) as convolution.

Moreover, it should be noted that the convolution is not defined correctly near the boundaries of the input signal.

This can be overcome if the convolution starts at an inner entry of the input signal such that the filter window is completely within the signal. The drawback of this procedure is that the resulting signal is smaller than the original one (Figure 2a). When it is mandatory to maintain the same size, it is possible to artificially add ghost entries to the input signal at the boundaries (padding). In this work, no padding has been considered.



**Figure 2.** (a) 1D single filter convolution, (b) 1D multi-filter convolution. The dimension of the 1D input signal is  $n$ , and each filter size is  $s$ .

Let us consider an input vector  $I$  (size  $n$ ) and a kernel  $K_{1,1}$  (size  $s$ ). Their convolution is represented by a new vector  $C$  (size  $n - s + 1$ ), as schematically shown in Figure 2a. It can be computed as follows:

$$C(i) = \sum_{u=1}^s I(i + u - 1)K_{1,1}(u). \tag{3}$$

The convolution (3) can be extended to the case of a collection of  $p$  filters with size  $s$ . The result of this multi-filter convolution is represented by a 2D matrix with size  $(n - s + 1) \times p$ , as schematically represented in Figure 2b, where each column  $C_j(i)$  is

$$C_j(i) = \sum_{u=1}^s I(i + u - 1)K_{1,j}(u) \tag{4}$$

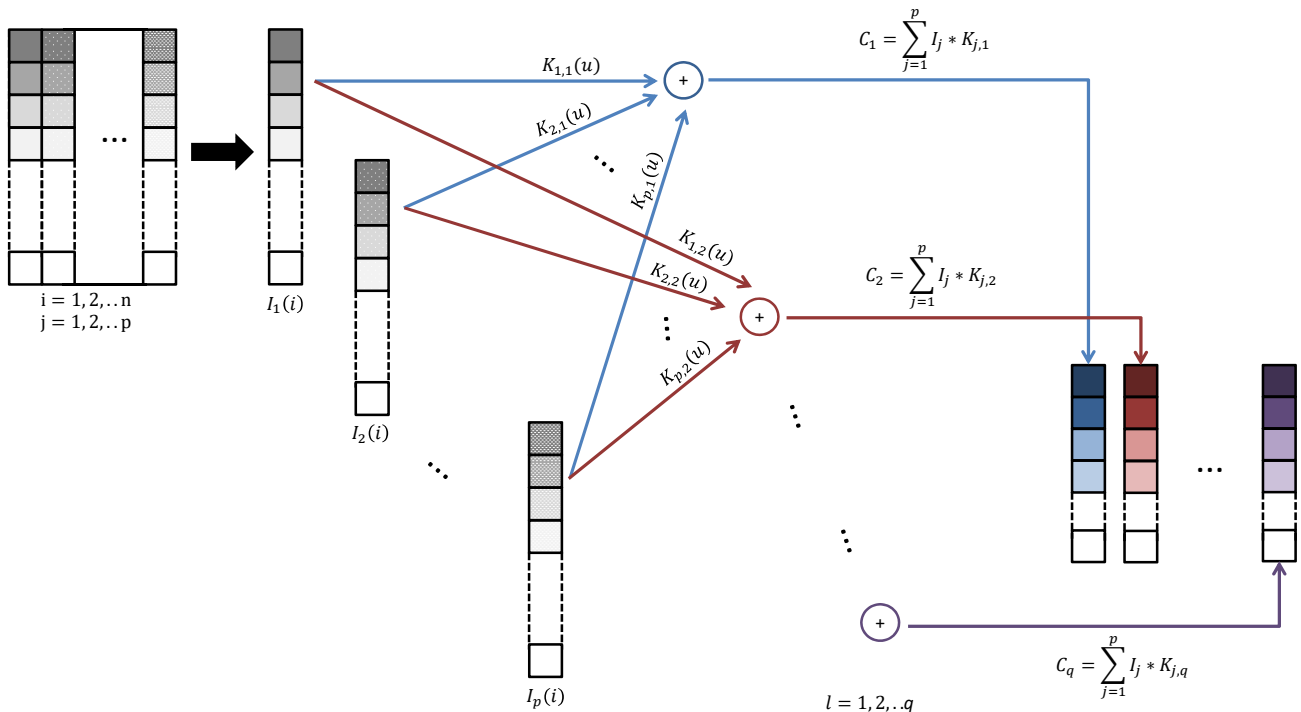
and  $p$  represents the number of filters.

Moreover, the convolution definition can be further extended to the case of an input array of  $p$  columns, as schematically depicted in Figure 3.

In this case, we can perform the convolution separately on each of the  $p$  columns  $I_j$  using a multi-filter represented by a third-order tensor (size  $s \times p \times q$ ). A 2D matrix (size  $(n - s + 1) \times q$ ) results from a convolution operation. This 2D matrix can be regarded as an array of  $q$  columns (Figure 3), in which each column ( $C_l(i)$ ) is obtained by summing up  $p$  convolutions:

$$C_l(i) = \sum_{j=1}^p I_j * K_{j,l}(u). \tag{5}$$

where  $l = 1, 2, \dots, q$ .



**Figure 3.** Convolution of an input signal  $p$  columns, each of length  $n$ , and a multi-filter represented by a 3D matrix of size  $s \times p \times q$ , where the index  $u$  runs from 1 to the filter size  $s$ , and  $p \times q$  is the number of filters. This process results in a 2D matrix in which each column  $C_q$  is obtained by summing up  $p$  convolutions.

### 3. 1D CNN Architecture

CNNs typically perform a series of operations to extract features from an input signal. For simplicity, this analysis focuses on a model with two convolutional layers ( $CL_1, CL_2$ ) followed by a fully connected layer ( $FCL$ ), as indicated in the “Layer” column of Table 1.

**Table 1.** Description of the considered CNN. The “Layer” column describes the types of layers used sequentially, the “Operation” column indicates the type of operation performed at each layer, the “Filter” column describes the number and size of kernels used in each layer, and the “Output size” specifies the size of the signal obtained at the end of each layer. The illustration of this 1D CNN is schematized as an image in the Supplementary Materials (Figure S1).

Layer	Operation	Filter		Output Size
		N.	Size	
$CL_1$	Conv + Act	$p = 4$	$s^{(1)} = 3$	$C_\sigma^{(1)} = 16 \times 4$
	Pool	1	$SS^{(1)} = 2$	$S^{(1)} = 8 \times 4$
$CL_2$	Conv + Act	$q = 6$	$s^{(2)} = 3$	$C_\sigma^{(2)} = 6 \times 6$
	Pool	1	$SS^{(2)} = 2$	$S^{(2)} = 3 \times 6$
$FCL$	Flat	–	–	$f = 18 \times 1$
	Fully connected + Act	–	–	$2 \times 1$

In turn, each convolution layer comprises convolution (Conv), activation (Act), and pooling (Pool) operations, as detailed in the “Operation” column of Table 1. The general features of the CNN architecture in terms of sizes are summarised in the “Filter” and “Output size” columns of Table 1. In  $CL_1$ , the convolution is performed using a single-channel input ( $I$ ) with dimension  $n = 18$  and  $p = 4$  filters with dimension  $s^{(1)} = 3$ . A bias array is then added to the output, and the sigmoid activation function is applied. A new array ( $C_\sigma^{(1)}$ ) of  $p$  columns with  $n - s^{(1)} + 1 = 16$  dimensions is obtained. It is then down-sampled (using max or average-pooling) with a factor  $SS^{(1)} = 2$ , resulting in a new array ( $S^{(1)}$ ) of  $p$  columns with size 8.

In  $CL_2$ , the convolution operation is performed using  $S^{(1)}$  as the input signal and  $p \times q$  filters with  $q = 6$  and  $s^{(2)} = 3$  as the dimension of each filter.

Similar to  $CL_1$ , a bias array is added here, and an activation function ( $\sigma$ ) is applied. This results in an array ( $C_\sigma^{(2)}$ ) of  $q$  columns of length 6. It is down-sampled (using max or average-pooling) by a factor of  $SS^{(2)} = 2$ . The resulting  $S^{(2)}$  is then represented by an array of  $q$  columns of size 3.

Finally, the output of  $CL_2$  is flattened, obtaining a new vector ( $f$ ) with dimension 18. This latter vector represents the input for the FCL. The output of the 1D CNN is represented by an array with two dimensions. This implies that a weight matrix  $2 \times 18$  is required in the FCL.

### 4. Feed Forward Propagation

In  $CL_1$ , the convolution is performed using the input signal ( $I$ ) and a multi-filter ( $K_{1,j}^{(1)}$ ), resulting in an array containing  $p$  elements:

$$C^{(1)} = (C_1^{(1)}, C_2^{(1)}, \dots, C_p^{(1)}). \tag{6}$$

Each element in (6) is then represented by

$$C_j^{(1)}(i) = I(i) * K_{1,j}^{(1)}(u) = \sum_{u=1}^{s^{(1)}} I(i + u - 1) K_{1,j}^{(1)}(u) + b_j^{(1)} \tag{7}$$

where the superscript refers to the convolutional layer number,  $j$  runs from 1 to the number of filters employed in the layer ( $p = 4$ ),  $u$  runs from 1 to the filter size  $s^{(1)}$ , and  $b_j^{(1)}$  represents the bias. In this specific case, as indicated in Table 1,  $p = 4$ ,  $s^{(1)} = 3$ , and  $i$  ranges from 1 to 16.

An activation function ( $\sigma$ ) is applied to each element of  $C^{(1)}$  and the resulting output  $C_\sigma^{(1)}$  is then pooled.

The primary objective of a pooling operation is to reduce the spatial dimensions of the input for the subsequent convolutional layer. Similar to the convolution operation, pooling uses a sliding window or a specific region that moves in stride across the input, transforming the values into representative ones. This transformation is commonly achieved by taking either the average or the maximum value from the input values within the window. The former is called “average pooling”, while the latter is called “max pooling”.

After pooling, a new array  $S^{(1)}$  with  $p$  columns is obtained. Each column  $S_j^{(1)}$  is then represented by

$$\begin{cases} S_j^{(1)}(i) = \text{Max}\{C_{j,\sigma}^{(1)}(i_s)\}_{W=1,2,\dots,SS^{(1)}} & \text{Max} \\ S_j^{(1)}(i) = \frac{1}{SS^{(1)}} \sum_{W=1}^{SS^{(1)}} C_{j,\sigma}^{(1)}(i_s) & \text{Average} \end{cases} \quad (8)$$

where  $i_s = SS^{(1)}i - W + 1$ . In this specific case,  $i_s$  runs from 1 to 16, and with  $SS^{(1)} = 2$ ,  $i$  runs from 1 to 8. For the sake of notation convenience, henceforth,  $i$  will represent the index  $i_s$ .

In  $CL_2$ , the convolution is performed using  $S^{(1)}$  as the input signal and a multi-filter represented by a third-order tensor. Since this type of tensor refers to a three-dimensional array of numbers, for clarity, the representation of its 2D components is provided in Figure 4.

$$\begin{pmatrix} K_{11}^{(2)}(1) & K_{11}^{(2)}(2) & K_{11}^{(2)}(3) \\ K_{21}^{(2)}(1) & K_{21}^{(2)}(2) & K_{21}^{(2)}(3) \\ K_{31}^{(2)}(1) & K_{31}^{(2)}(2) & K_{31}^{(2)}(3) \\ K_{41}^{(2)}(1) & K_{41}^{(2)}(2) & K_{41}^{(2)}(3) \end{pmatrix}$$
  

$$\begin{pmatrix} K_{12}^{(2)}(1) & K_{12}^{(2)}(2) & K_{12}^{(2)}(3) \\ K_{22}^{(2)}(1) & K_{22}^{(2)}(2) & K_{22}^{(2)}(3) \\ K_{32}^{(2)}(1) & K_{32}^{(2)}(2) & K_{32}^{(2)}(3) \\ K_{42}^{(2)}(1) & K_{42}^{(2)}(2) & K_{42}^{(2)}(3) \end{pmatrix}$$
  

$$\vdots$$
  

$$\begin{pmatrix} K_{16}^{(2)}(1) & K_{16}^{(2)}(2) & K_{16}^{(2)}(3) \\ K_{26}^{(2)}(1) & K_{26}^{(2)}(2) & K_{26}^{(2)}(3) \\ K_{36}^{(2)}(1) & K_{36}^{(2)}(2) & K_{36}^{(2)}(3) \\ K_{46}^{(2)}(1) & K_{46}^{(2)}(2) & K_{46}^{(2)}(3) \end{pmatrix}$$

**Figure 4.** The filter  $K^{(2)}$  is a third-order tensor with  $s^{(2)} \times p \times q$  dimension, where  $s^{(2)} = 3$ ,  $p = 4$  and  $q = 6$ . The 2D components of  $K^{(2)}$  are represented separately.



An array of  $q$  columns is then obtained, and each column is represented by

$$C_l^{(2)} = \sum_{j=1}^p S_j^{(1)} * K_{j,l}^{(2)} + b_l^{(2)} \tag{9}$$

In this specific case,  $p = 4$ ,  $l$  runs from 1 to  $q = 6$ , and  $i$  from 1 to  $8 - 3 + 1 = 6$ .

An activation function is then applied to each element in  $C^{(2)}$ , and the resulting output  $C_\sigma^{(2)}$  is then pooled. A new array  $S^{(2)}$  with  $q$  columns is obtained, in which each column  $S_j^{(2)}$  is then represented by

$$\begin{cases} S_j^{(2)}(i) = \text{Max}\{C_{j,\sigma}^{(2)}(i_s)\}_{w=1,2,\dots,SS^{(2)}} & \text{Max} \\ \text{or} \\ S_j^{(2)}(i) = \frac{1}{SS^{(2)}} \sum_{w=1}^{SS^{(2)}} C_{j,\sigma}^{(2)}(i_s) & \text{Average} \end{cases} \tag{10}$$

where  $i_s = SS^{(2)}i - w + 1$ . In  $CL_2$ ,  $i_s$  runs from 1 to 6, and with  $SS^{(2)} = 2$   $i$  runs from 1 to 3. Again, for convenience of notation, in the following,  $i$  will denote the index  $i_s$ .

In the FCL, the first operation to be performed is represented by a flattening. It involves the 2D output generated at the end of the pooling step in  $CL_2$  ( $S^{(2)}$ ) and transforms it into a 1D vector. Generally, this operation can be performed by reshaping in row-major or column-major order. In the case under consideration,  $S^{(2)}$  is a  $3 \times 6$  matrix. It can be reshaped to form a new 1D vector as follows:

$$\begin{cases} f(i + m(j - 1)) = S_j^{(2)}(i) & \text{column-major} \\ f(j + n(i - 1)) = S_j^{(2)}(i) & \text{row-major} \end{cases} \tag{11}$$

where the row index ( $i$ ) of  $S^{(2)}$  runs from 1 to  $m = 3$ , and the column index  $j$  from 1 to  $n = 6$ . In this specific case, the new vector ( $f$ ) has  $m \times n = 18$ .

Once the vector  $f$  has been obtained, it can be fed to the fully connected network. Then, each element of the output vector  $y_{out} = \sigma(W \times f + b)$  can be written as

$$y_{out}(i) = \sigma\left(\sum_{j=1}^{18} W(i, j)f(j) + b(i)\right) \tag{12}$$

where  $\sigma$  is an activation function ( $\sigma$ ),  $W$  is the weight matrix  $2 \times 18$ , and  $b$  a bias vector  $2 \times 1$ .

### 5. Backpropagation

The training process of the network comprises both feedforward and backpropagation, as schematically depicted in Figure 5.

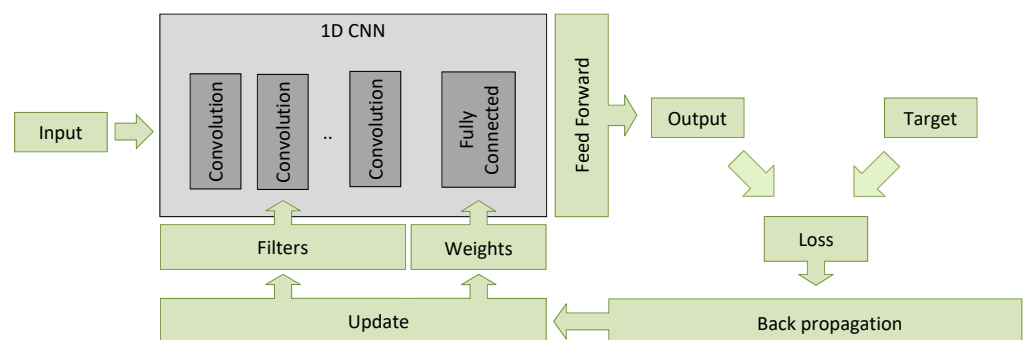


Figure 5. Schematization of the training process.

Backpropagation aims to minimize a loss function by updating the network's parameters (e.g., weights and biases) so that the network can correctly map arbitrary inputs to outputs. The choice of loss function depends on the specific task at hand, as different tasks require different loss functions to effectively measure and optimize model performance.

Our literature review indicates that regression and classification are among the most commonly used tasks. Typically, a regression task aims to predict the value of a continuous variable, such as a price or probability. For instance, a 1D CNN can be used in regression tasks to predict the concentration of a specific compound in a sample based on its absorption spectrum. In this example, the input 1D signals are represented by absorbance values recorded at various wavelengths (spectra). Additionally, the same spectral data can be employed for classification tasks by making use of different loss function. In this case, the goal is to categorize different chemical compounds into predefined categories.

While the underlying architecture of 1D CNNs for both regression and classification tasks involves convolutional layers to extract features from sequential data, the key differences lie in the output layer configuration, the choice of activation functions, the loss functions used for training, and the evaluation metrics. For regression, the output is a continuous value and the model is trained to minimize the error between predicted and actual values. For classification, the output is categorical, and the model is trained to correctly classify inputs into one of several classes.

Backpropagation is the process used in training neural networks, where the error (or loss) is propagated backward through the layers of the network, starting from the output layer and moving toward the input layer. Its main purpose is to calculate the gradients (partial derivatives) of the loss function with respect to each parameter (weight) in the network. These gradients indicate how much each parameter needs to change to reduce the loss. The derivatives can be efficiently computed using the chain rule and can be used in the gradient descent algorithm to update the network's parameters in a way that minimizes the loss function.

In this section, the gradients of each parameter are computed from end to start (e.g., from FCL to  $CL_1$ ). In regression problems, the loss function ( $L$ ) is computed by comparing the actual target  $y_t$  with the predicted output values  $y_{out}$ . One of the most popular loss functions is represented by the mean squared error (MSE):

$$L = \frac{1}{N} \sum_{i=1}^N (y_{out}(i) - y_t(i))^2 \quad (13)$$

where  $i$  runs from 1 to the dimension of the output, in this case  $N = 2$ .

Let us begin with the FCL. In this case, the parameters to be updated are the weight matrix  $W$  and the bias  $b$ . The first gradient to be calculated is  $\Delta W$ , with each entry represented by

$$\begin{aligned} \Delta W(i, j) &= \frac{\partial L}{\partial W(i, j)} \\ &= \frac{\partial L}{\partial y_{out}(i)} \cdot \frac{\partial y_{out}(i)}{\partial W(i, j)} \\ &= (y_{out}(i) - y_t(i)) \cdot \frac{\partial}{\partial W(i, j)} \sigma \left( \sum_{j=1}^{18} W(i, j) f(j) + b(i) \right) \end{aligned} \quad (14)$$

where “ $\cdot$ ” refers to the element-wise product. The last derivative in Equation (14) can be calculated once an activation function is provided. Discussing the advantages and disadvantages of using a specific activation function over another is beyond the scope of this work. For this purpose, we refer the curiosity of the reader to the existing literature on this topic [7,125–129].

In this output layer, the sigmoid activation function has been considered [130], while

other functions will be discussed in Appendix A.

Defining  $\Delta y_{out}$  as a vector  $2 \times 1$  vector

$$\Delta y_{out}(i) = (y_{out}(i) - y_t(i)) \cdot y_{out}(i) \cdot (1 - y_{out}(i)) \tag{15}$$

the gradient  $\Delta W$  can be rewritten as

$$\Delta W = \Delta y_{out} \times f^T. \tag{16}$$

where “ $\times$ ” refers to the matrix product, and  $f^T$  represents the transpose of the vector  $f$ , which is a  $1 \times 18$  vector.

The same calculation can be applied for the bias gradient:

$$\begin{aligned} \Delta b(i) &= \frac{\partial L}{\partial b(i)} \\ &= \frac{\partial L}{\partial y_{out}(i)} \cdot \frac{\partial y_{out}(i)}{\partial b(i)} \\ &= (y_{out}(i) - y_t(i)) \cdot \frac{\partial}{\partial b(i)} \sigma \left( \sum_{j=1}^{18} W(i, j) f(j) + b(i) \right) \\ &= (y_{out}(i) - y_t(i)) \cdot y_{out}(i) \cdot (1 - y_{out}(i)) \\ &= \Delta y_{out}(i). \end{aligned} \tag{17}$$

Before updating the parameters in  $CL_2$ , it is necessary to calculate the gradient of the vector  $f$ . Each entry of  $\Delta f$  is then represented by

$$\begin{aligned} \Delta f(j) &= \frac{\partial L}{\partial f(j)} \\ &= \sum_{i=1}^2 \frac{\partial L}{\partial y_{out}(i)} \cdot \frac{\partial y_{out}(i)}{\partial f(j)} \\ &= \sum_{i=1}^2 (y_{out}(i) - y_t(i)) \cdot y_{out}(i) \cdot (1 - y_{out}(i)) \cdot W(i, j) \\ &= \sum_{i=1}^2 \Delta y_{out}(i) W(i, j) \end{aligned} \tag{18}$$

and  $\Delta f$  can now be written as

$$\Delta f = W^T \times \Delta y_{out}. \tag{19}$$

During the feedforward step, the vector  $f$  results from  $S^{(2)}$  reshaping. Here, in the backpropagation step, the gradient  $\Delta f$  is reshaped back to obtain  $\Delta S^{(2)}$ , as schematically depicted in Figure 6.

In the  $CL_2$ , an un-pooling operation should be performed to up-sample the compressed data  $\Delta S^{(2)}$  and obtain  $\Delta C_\sigma^{(2)}$ . To avoid confusion, great attention is now needed. If average up-sampling was used in feedforward, the backpropagation error is multiplied by  $1/SS^{(2)}$  and uniformly assigned to all the elements in the pooling block (Figure 7).

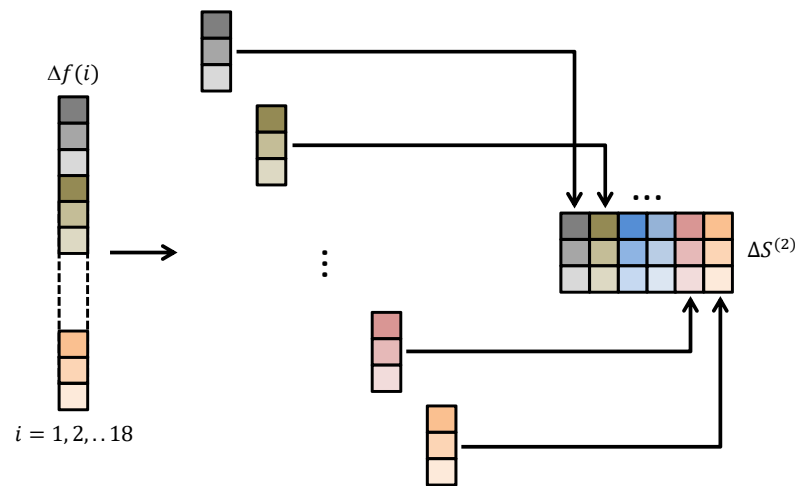


Figure 6. The vector  $f$  ( $18 \times 1$ ) is reshaped back in order to obtain  $\Delta S^{(2)}$  ( $3 \times 6$ ).

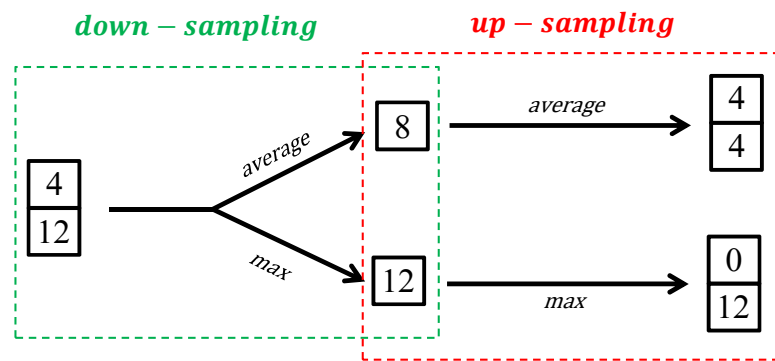


Figure 7. Example of an up-sampling operation. The input is the  $2 \times 1$  array on the left. Down-sampling produces a 1D array based on the average or maximum of the input. During up-sampling, the 1D array is split into a new  $2 \times 1$  array. If averaging is used, the output is filled by evenly distributing the output value from down-sampling, in this case, dividing by 2. If the max operation is used, the output value is placed where the original maximum value occurred.

In this case, each column of  $\Delta C^{(2)}$  can be written as

$$\Delta C_{j\sigma}^{(2)}(i_u) = \frac{1}{SS^{(2)}} \Delta S_j^{(2)} \left( \lceil i_u / SS^{(2)} \rceil \right) \tag{20}$$

where the index  $i$  of  $\Delta S_j^{(2)}$  runs from 1 to  $n = 3$ ,  $i_u$  runs from 1 to  $6 = SS^{(2)}n$ , and finally  $\lceil x \rceil$  represents the ceiling function. As in previous instances, in the following,  $i$  will denote the index  $i_u$ .

On the other hand, if max-pooling was performed during the feedforward step, the error is assigned to the location it originated from, and zero is assigned to all the other elements. This implies that during the feedforward step, the index of the “winning unit” should be stored and reused in the backpropagation step. This aspect may not be evident when using pre-packaged solutions but should be taken into account when developing new CNN libraries.

Once  $\Delta C_{\sigma}^{(2)}$  has been computed, it is possible to update the filter value in  $CL_2$ . This means calculating each element of the third-order tensor  $\Delta K^{(2)}$ , with  $j = 1, \dots, p = 4$ ,  $l = 1, \dots, q = 6$ , and  $u = 1, \dots, 3$ , we obtain

$$\begin{aligned}
 \Delta K_{j,l}^{(2)}(u) &= \frac{\partial L}{\partial K_{j,l}^{(2)}(u)} \\
 &= \sum_{i=1}^6 \frac{\partial L}{\partial C_{l,\sigma}^{(2)}(i)} \cdot \frac{\partial C_{l,\sigma}^{(2)}(i)}{\partial K_{j,l}^{(2)}(u)} \\
 &= \sum_{i=1}^6 \Delta C_{l,\sigma}^{(2)}(i) \cdot \frac{\partial \sigma(C_l^{(2)}(i))}{\partial K_{j,l}^{(2)}(u)} \\
 &= \sum_{i=1}^6 \Delta C_{l,\sigma}^{(2)}(i) \cdot \frac{\partial \sigma(C_l^{(2)}(i))}{\partial C_l^{(2)}(i)} \cdot \frac{\partial C_l^{(2)}(i)}{\partial K_{j,l}^{(2)}(u)} \\
 &= \sum_{i=1}^6 \Delta C_{l,\sigma}^{(2)}(i) \cdot C_{l,\sigma}^{(2)}(i) \cdot (1 - C_{l,\sigma}^{(2)}(i)) \cdot \frac{\partial C_l^{(2)}(i)}{\partial K_{j,l}^{(2)}(u)} \\
 &= \sum_{i=1}^6 \Delta C_l^{(2)}(i) \cdot \frac{\partial C_l^{(2)}(i)}{\partial K_{j,l}^{(2)}(u)}
 \end{aligned} \tag{21}$$

where  $\Delta C_l^{(2)} = \Delta C_{l,\sigma}^{(2)} \cdot C_{l,\sigma}^{(2)} \cdot (1 - C_{l,\sigma}^{(2)})$  has been obtained, considering sigmoid as the activation function ( $\sigma$ ). In these hidden layers, other activation functions may be used, these cases will be discussed in the Appendix A.

Recalling Equations (3) and (9), the last derivative in (21) can be written as

$$\begin{aligned}
 \frac{\partial C_l^{(2)}}{\partial K_{j,l}^{(2)}} &= \frac{\partial}{\partial K_{j,l}^{(2)}} \left( \sum_{j=1}^{n=4} S_j^{(1)} * K_{j,l}^{(2)} + b_l^{(2)} \right) \\
 &= \frac{\partial}{\partial K_{j,l}^{(2)}} \left( S_1^{(1)} * K_{1,l}^{(2)} + S_2^{(1)} * K_{2,l}^{(2)} + \dots + S_n^{(1)} * K_{n,l}^{(2)} + b_l^{(2)} \right) \\
 &= \frac{\partial}{\partial K_{j,l}^{(2)}} \left( S_j^{(1)} * K_{j,l}^{(2)} \right)
 \end{aligned} \tag{22}$$

and since only  $S_j^{(1)} \cdot K_{j,l}^{(2)}$  is relevant for the computation, we obtain

$$\frac{\partial C_l^{(2)}(i)}{\partial K_{j,l}^{(2)}(u)} = S_j^{(1)}(i + u - 1). \tag{23}$$

Each element of the error  $\Delta K_{j,l}^{(2)}$  can now be written as

$$\Delta K_{j,l}^{(2)}(u) = \sum_{i=1}^6 \Delta C_l^{(2)}(i) S_j^{(1)}(i + u - 1) \tag{24}$$

and then the error is

$$\Delta K_{j,l}^{(2)} = S_j^{(1)} * \Delta C_l^{(2)} \tag{25}$$

where “ $*$ ” represents the convolution operation. In the present case,  $S_j^{(1)}$  and  $\Delta C_l^{(2)}$  are  $8 \times 1$  and  $6 \times 1$  1D columns, respectively. The convolution in Equation (23) yields a  $3 \times 1$  vector, corresponding to the filter dimension selected in  $CL_2$  ( $s^{(2)}$ ).

Now, great attention should be paid when using Equation (25). In the literature, there are many examples of similar formulae in which the vector  $S_j^{(1)}$  is used in reverse [120], without providing details. The reversing operation is required only when the convolution definition in (2) is used.

Following a similar procedure, it is possible to compute the bias  $b_l^{(2)}$  error. Since  $C_l^{(2)}$  is linear with the bias,  $\frac{\partial C_l^{(2)}}{\partial b_l^{(2)}} = 1$ , then the bias error is computed as follows:

$$\begin{aligned}
 \Delta b_l^{(2)} &= \frac{\partial L}{\partial b_l^{(2)}} \\
 &= \sum_{i=1}^6 \frac{\partial L}{\partial C_{l,\sigma}^{(2)}(i)} \cdot \frac{\partial C_{l,\sigma}^{(2)}(i)}{\partial b_l^{(2)}} \\
 &= \sum_{i=1}^6 \Delta C_{l,\sigma}^{(2)}(i) \cdot \frac{\partial \sigma(C_l^{(2)}(i))}{\partial b_l^{(2)}} \\
 &= \sum_{i=1}^6 \Delta C_{l,\sigma}^{(2)}(i) \cdot \frac{\partial \sigma(C_l^{(2)}(i))}{\partial C_l^{(2)}(i)} \cdot \frac{\partial C_l^{(2)}(i)}{\partial b_l^{(2)}} \\
 &= \sum_{i=1}^6 \Delta C_{l,\sigma}^{(2)}(i) \cdot C_{l,\sigma}^{(2)}(i)(1 - C_{l,\sigma}^{(2)}(i)) \\
 &= \sum_{i=1}^6 \Delta C_l^{(2)}(i).
 \end{aligned}
 \tag{26}$$

Let us now enter the  $CL_1$ . The object here is to update filters  $K_{1,j}^{(1)}$  and bias  $b_j^{(1)}$ . To achieve this, the error  $\Delta S_j^{(1)}$  must be calculated first.

Using Equations (1) and (9),  $C_l^{(2)}(i)$  can be expressed as a function of  $S_j^{(2)}(i + u - 1)$ . Introducing a new index  $I = i + u - 1$ , we find that  $C_l^{(2)}(I - u + 1)$  can be represented as a function of  $S_j^{(2)}(I)$ . Moreover, for simplicity, if we denote  $I = i$ ,  $C_l^{(2)}(i - u + 1)$  can be written as a function of  $S_j^{(2)}(i)$ . This result is utilized when computing each element of the error  $\Delta S_j^{(1)}$  as follows:

$$\begin{aligned}
 \Delta S_j^{(1)}(i) &= \frac{\partial L}{\partial S_j^{(1)}(i)} \\
 &= \sum_{l=1}^{q=6} \sum_{u=1}^{s^{(2)}} \frac{\partial L}{\partial C_l^{(2)}(i - u + 1)} \cdot \frac{\partial C_l^{(2)}(i - u + 1)}{\partial S_j^{(1)}(i)} \\
 &= \sum_{l=1}^{q=6} \sum_{u=1}^{s^{(2)}} \Delta C_l^{(2)}(i - u + 1) \cdot \frac{\partial C_l^{(2)}(i - u + 1)}{\partial S_j^{(1)}(i)}
 \end{aligned}
 \tag{27}$$

where  $s^{(2)} = 3$  is the filter size used in  $CL_2$ .

The partial derivative in Equation (27) can be computed by making use of Equation (9) evaluated in  $i - u + 1$ . Each element of the error  $\Delta S_j^{(1)}$  is then

$$\begin{aligned}
 \Delta S_j^{(1)}(i) &= \sum_{l=1}^{q=6} \sum_{u=1}^{s^{(2)}} \Delta C_l^{(2)}(i - u + 1) \cdot K_{j,l}^{(2)}(u) \\
 &= \sum_{l=1}^{q=6} \sum_{u=1}^{s^{(2)}} \Delta C_l^{(2)}(i + (-u) + 1) \cdot K_{j,l}^{(2)}(-(-u))
 \end{aligned}
 \tag{28}$$

and finally, the error can be rewritten as

$$\Delta S_j^{(1)} = \sum_{l=1}^{q=6} \Delta C_l^{(2)} * K_{j,l}^{(2)} \tag{29}$$

where  $K_{j,l}^{(2)}$  is obtained by reversing the order of the elements in  $K_{j,l}^{(1)}$ .

Similar to the approach used in  $CL_2$  for average pooling, here in  $CL_1$  we can write

$$\Delta C_{j\sigma}^{(1)}(i_u) = \frac{1}{SS^{(1)}} \Delta S_j^{(1)} \left( \lceil i_u / SS^{(1)} \rceil \right) \tag{30}$$

where the index  $i$  of  $\Delta S_j^{(1)}$  runs from 1 to  $n = 8$ ,  $i_u$  runs from 1 to  $16 = SS^{(1)}n$ , and finally,  $\lceil x \rceil$  represents the ceiling function. As in previous instances, in the following,  $i$  will denote the index  $i_u$ .

Finally, we can calculate each element of the error  $\Delta K_{1,j}^{(1)}$  as follows:

$$\begin{aligned} \Delta K_{1,j}^{(1)}(u) &= \frac{\partial L}{\partial K_{1,j}^{(1)}(u)} \\ &= \sum_{i=1}^{16} \frac{\partial L}{\partial C_{j,\sigma}^{(1)}(i)} \frac{\partial C_{j,\sigma}^{(1)}(i)}{\partial K_{1,j}^{(1)}(u)} \\ &= \sum_{i=1}^{16} \Delta C_{j,\sigma}^{(1)}(i) \cdot C_{j,\sigma}^{(1)}(i) (1 - C_{j,\sigma}^{(1)}(i)) \cdot I(i + u - 1) \\ &= \sum_{i=1}^{16} \Delta C_j^{(1)}(i) \cdot I(i + u - 1) \end{aligned} \tag{31}$$

and then the error is

$$\Delta K_{1,j}^{(1)} = I * \Delta C_j^{(1)} \tag{32}$$

where  $\Delta C_j^{(1)} = \Delta C_{j,\sigma}^{(1)} C_{j,\sigma}^{(1)} (1 - C_{j,\sigma}^{(1)})$  has been obtained considering sigmoid as the activation function ( $\sigma$ ). Again, other activation functions may be used in these hidden layers. These cases will be discussed in the Appendix A.

Analogously, the bias error is

$$\begin{aligned} \Delta b_j^{(1)} &= \frac{\partial L}{\partial b_j^{(1)}} \\ &= \sum_{i=1}^{16} \frac{\partial L}{\partial C_{j,\sigma}^{(1)}(i)} \frac{\partial C_{j,\sigma}^{(1)}(i)}{\partial b_j^{(1)}} \\ &= \sum_{i=1}^{16} \Delta C_{j,\sigma}^{(1)}(i) \cdot C_{j,\sigma}^{(1)}(i) (1 - C_{j,\sigma}^{(1)}(i)) \\ &= \sum_{i=1}^{16} \Delta C_j^{(1)}(i). \end{aligned} \tag{33}$$

where the sigmoid has been used as the activation function, and the substitution  $\Delta C_l^{(2)} = \Delta C_{l,\sigma}^{(2)} \cdot C_{l,\sigma}^{(2)} \cdot (1 - C_{l,\sigma}^{(2)})$  introduced above.

### 6. Other Activation and Loss Functions

In addition to the sigmoid as an activation function, which was considered in Section 5, here, we aim to extend the calculation to also consider the hyperbolic tangent (tanh) [131–134],

logarithmic hyperbolic cosine (log cosh) [135], and Rectified Linear Unit (ReLU) [136], including some of its variants [129,137,138].

Moreover, along with MSE, other loss functions are explored for both regression and classification problems. To provide the correct expressions for the different cases, in backpropagation, what needs to be modified are only the expressions for  $\Delta y_{out}(i)$ ,  $\Delta C_1^{(2)}(i)$ , and  $\Delta C_j^{(1)}(i)$ . Once computed, they can be used in Equations (15)–(33). Generally, the choice of the activation function depends on which layer you are dealing with [139].

Although ReLU is commonly preferred in hidden layers (here  $CL_1$  and  $CL_2$ ), there are examples of its use in output layers [140] when the response variable is continuous and greater than zero [141].

In this section, we assume that the ReLU activation function is applied exclusively in hidden layers, while the other activation functions are used in both the output and hidden layers. Let us start by discussing the regression task. The various configurations explored in terms of activation and loss functions for this task are summarized in the Tables 2 and 3.

The expression for  $\Delta y_{out}(i)$  depends on both the loss and the activation functions considered in the output layer. When MSE is used with the sigmoid, the valid expression is the one provided in Section 5. Instead, when tanh is used, we have

$$\Delta y_{out}(i) = (y_{out}(i) - y_t(i)) \cdot (1 - y_{out}^2(i)). \tag{34}$$

where the first factor comes from the derivative of MSE, and the second one from the derivative of tanh, which is  $1 - \tanh^2$ .

Another loss function commonly used in regression problems is represented by the log cosh function

$$L = \frac{1}{N} \sum_{i=1}^N (\log(\cosh(y_{out}(i) - y_t(i)))) \tag{35}$$

where  $i$  runs from 1 to the dimension of the output (here,  $N = 2$ ). With log cosh as loss function, the expression (34) becomes

$$\Delta y_{out}(i) = \frac{1}{2} \tanh(y_{out}(i) - y_t(i)) y_{out}(i) \cdot (1 - y_{out}(i)) \tag{36}$$

for sigmoid activation function in the output layer, and

$$\Delta y_{out}(i) = \frac{1}{2} \tanh(y_{out}(i) - y_t(i)) \cdot (1 - y_{out}^2(i)) \tag{37}$$

for tanh activation in the output layer. As in Equations (34)–(36), the first factor comes from the derivative of the loss function, and the second comes from that of the activation function.

**Table 2.** Expressions for  $\Delta y_{out}$  corresponding to different loss and activation functions in the output layer.

Loss Function	Activation Function in Output Layer	$\Delta y_{out}$
MSE	sigmoid	(15)
MSE	tanh	(34)
log cosh	sigmoid	(36)
log cosh	tanh	(37)



**Table 3.** Expressions for  $\Delta C^{(n)}$  corresponding to different activation functions in hidden layers.

Activation Function in Hidden Layer	$\Delta C^{(n)}$
sigmoid	(20),(30)
tanh	(38)
ReLU	(40)
ELU	(43)
Leaky ReLU	(44)

The different activation functions available for hidden layers have an impact on both  $\Delta C_l^{(2)}(i)$  and  $\Delta C_j^{(1)}(i)$  expressions. When the sigmoid is used, the valid expressions are those provided in Section 5. Instead, when the tanh activation function is used, we have

$$\Delta C_l^{(n)}(i) = \Delta C_{l,\sigma}^{(n)}(i) \left( 1 - \left( C_{l,\sigma}^{(n)}(i) \right)^2 \right) \tag{38}$$

where  $n = 1, 2$  represents the hidden layer number.

Moreover, with the ReLU activation function in hidden layers defined as follows:

$$ReLU(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{39}$$

Equation (38) can be rewritten:

$$\Delta C_l^{(n)}(i) = \Delta C_{l,\sigma}^{(n)}(i) \begin{cases} 1 & C_l^{(n)}(i) \geq 0 \\ 0 & C_l^{(n)}(i) < 0 \end{cases} \tag{40}$$

where  $n = 1, 2$  represents the hidden layer number.

Among the possible variants of the ReLU activation function, we can consider ELU (Exponential Linear Unit):

$$ELU(x) = \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases} \tag{41}$$

and Leaky ReLU

$$LeakyReLU(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x \leq 0 \end{cases} \tag{42}$$

with  $\alpha$  a parameter that might be considered as trainable.

The expression (38) is now

$$\Delta C_l^{(n)}(i) = \Delta C_{l,\sigma}^{(n)}(i) \begin{cases} 1 & C_l^{(n)}(i) > 0 \\ \alpha e^{C_l^{(n)}(i)} & C_l^{(n)}(i) \leq 0 \end{cases} \tag{43}$$

for ELU, and

$$\Delta C_l^{(n)}(i) = \Delta C_{l,\sigma}^{(n)}(i) \begin{cases} 1 & C_l^{(n)}(i) \geq 0 \\ \alpha & C_l^{(n)}(i) < 0 \end{cases} \tag{44}$$

for Leaky ReLU.

Let us shift our focus to classification tasks. In these scenarios, the output layer provides probabilities for decision-making when classifying input data. Essentially, the output is presented as a list of probabilities for different potential labels. When calculating the error, the target output  $y_t(i)$  is represented by a one-hot vector that has 1 at one index

and 0 at all other indices. Consequently, this implies that  $\sum_{k=1}^N y_t(k) = 1$ . Since the output of the output layer may assume both positive and negative real values, the output  $y_{out}(i)$  needs to be transformed into probability-like values. This can be achieved using the Softmax function:

$$P_i = \frac{e^{y_{out}(i)}}{\sum_{k=1}^N e^{y_{out}(k)}} \tag{45}$$

where  $0 \leq P_i \leq 1$  and  $\sum_{i=1}^N P_i = 1$ . The derivative of Equation (43) can be easily calculated as follows:

$$\frac{\partial P_k}{\partial y_{out}(i)} = \begin{cases} -P_k P_i & k \neq i \\ P_k(1 - P_i) & k = i. \end{cases} \tag{46}$$

When using the Softmax as the output activation function, the loss function that is generally associated is represented by the cross-entropy, given by

$$L = - \sum_{i=1}^N y_t(i) \ln(P_i). \tag{47}$$

The expression for  $\Delta y_{out}(i)$  can be derived after calculating the derivative of the loss, which is obtained using the derivative of Equation (44):

$$\begin{aligned} \frac{\partial L}{\partial y_{out}(i)} &= \sum_{k=1}^N \frac{\partial L}{\partial P_k} \frac{\partial P_k}{\partial y_{out}(i)} \\ &= \sum_{k=1}^N \left( \frac{y_t(k)}{P_k} P_k P_i \right) - \frac{y_t(i)}{P_i} P_i \\ &= P_i - y_t(i). \end{aligned} \tag{48}$$

### 7. Conclusions

This tutorial thoroughly covers the theoretical aspects of feedforward and backpropagation in 1D CNNs, providing a step-by-step guide. It points out existing relationships with linear algebra, statistics, and optimization, addressing gaps identified in our literature review. The provided results consider both regression and classification tasks and include some of the most widely used activation functions. Its goal is to support the development of practical applications using these mathematical concepts. Moreover, the tutorial highlights potential challenges and pitfalls commonly encountered when implementing new library packages for 1D CNN in languages other than the available open-source ones. By bridging theoretical foundations with practical insights, this tutorial serves as a comprehensive guide for both beginners and experienced professionals aiming to deepen their understanding of 1D CNNs. In conclusion, this tutorial not only enhances comprehension of 1D CNNs but also equips beginners with the confidence to apply these techniques effectively. Future research directions might include exploring advanced architectures, integrating novel activation functions, and leveraging emerging optimization algorithms to further improve the performance and applicability of 1D CNNs.

**Supplementary Materials:** The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/app14188500/s1>, Figure S1: illustration of the 1D CNN considered, the caption is ‘‘Schematic representation of the 1D CNN studied in the tutorial’’.

**Author Contributions:** Conceptualization, I.C.; methodology, I.C. and A.R.; formal analysis, I.C.; data curation, I.C. and A.R.; writing original draft preparation, I.C. and A.R.; writing review and editing, I.C. and A.R. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The authors declare that the data supporting the findings of this study are available from the corresponding author on request.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

DL	Deep Learning
DBN	Deep Belief Network
DMC	Dry Matter Content
DNN	Deep Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural network
GAN	Generative Adversarial Network
DBN	Deep Belief Network
EEG	Electroencephalogram
ECG	Electrocardiogram
ReLU	Rectified Linear Unit
FCL	Fully Connected Layer
MSE	Mean Squared Error
ELU	Exponential Rectified Unit

## Appendix A

In this section, all the derivatives used for backpropagation are calculated by making use of the rule. Let us consider a vector  $y$  of length  $m$ , which is calculated by forming the product of a matrix  $A$  with  $m$  rows and  $n$  columns:

$$\begin{aligned} y_1 &= a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \\ y_2 &= a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \\ &\vdots \\ y_m &= a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \end{aligned} \quad (A1)$$

in other words, each element of the vector  $y$  can be written as follows:

$$y_i = \sum_{j=1}^n a_{ij}x_j. \quad (A2)$$

The derivative of  $y$  with respect to  $x$  requires calculating the partial derivative of each component of  $y$  with respect to each component of  $x$ :

$$\nabla_x y = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \dots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \dots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} = A^T. \quad (A3)$$

The derivative of  $y$  with respect to  $A$  is calculated considering that  $\frac{\partial y_i}{\partial a_{jk}}$  vanishes for  $i \neq j$ , we obtain

$$\nabla_A y = x^T. \quad (A4)$$

Let us now consider the vector  $z$  as a function of  $y = Ax$ . In this case, the derivative with respect to  $x$  is represented by a vector  $n \times 1$  and can be calculated as follows:

$$\nabla_x z = \nabla_x f(Ax) = \begin{bmatrix} \frac{\partial f(Ax)}{\partial x_1} \\ \frac{\partial f(Ax)}{\partial x_2} \\ \vdots \\ \frac{\partial f(Ax)}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f(y_1, y_2, \dots, y_m)}{\partial x_1} \\ \frac{\partial f(y_1, y_2, \dots, y_m)}{\partial x_2} \\ \vdots \\ \frac{\partial f(y_1, y_2, \dots, y_m)}{\partial x_n} \end{bmatrix}. \tag{A5}$$

Making use of the chain rule we obtain

$$\nabla_x f(Ax) = \begin{bmatrix} \frac{\partial f}{\partial y_1} \frac{\partial y_1}{\partial x_1} + \frac{\partial f}{\partial y_2} \frac{\partial y_2}{\partial x_1} + \dots + \frac{\partial f}{\partial y_m} \frac{\partial y_m}{\partial x_1} \\ \frac{\partial f}{\partial y_1} \frac{\partial y_1}{\partial x_2} + \frac{\partial f}{\partial y_2} \frac{\partial y_2}{\partial x_2} + \dots + \frac{\partial f}{\partial y_m} \frac{\partial y_m}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial y_1} \frac{\partial y_1}{\partial x_n} + \frac{\partial f}{\partial y_2} \frac{\partial y_2}{\partial x_n} + \dots + \frac{\partial f}{\partial y_m} \frac{\partial y_m}{\partial x_n} \end{bmatrix}. \tag{A6}$$

Each entry of this matrix  $\left(\frac{\partial y_j}{\partial x_k}\right)$  is  $a_{jk}$ ; then Equation (A6) can be rewritten as

$$\nabla_x f(Ax) = A^T \frac{\partial f}{\partial y} \tag{A7}$$

where  $A^T$  is the transpose of the  $A$  matrix and

$$\frac{\partial f}{\partial y} = \begin{bmatrix} \frac{\partial f}{\partial y_1} \\ \frac{\partial f}{\partial y_2} \\ \vdots \\ \frac{\partial f}{\partial y_m} \end{bmatrix}. \tag{A8}$$

Now, we are interested in calculating the derivative with respect to  $A$ :

$$\nabla_A f(Ax) = \begin{bmatrix} \frac{\partial f(Ax)}{\partial a_{11}} & \frac{\partial f(Ax)}{\partial a_{12}} & \dots & \frac{\partial f(Ax)}{\partial a_{1n}} \\ \frac{\partial f(Ax)}{\partial a_{21}} & \frac{\partial f(Ax)}{\partial a_{22}} & \dots & \frac{\partial f(Ax)}{\partial a_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(Ax)}{\partial a_{m1}} & \frac{\partial f(Ax)}{\partial a_{m2}} & \dots & \frac{\partial f(Ax)}{\partial a_{mn}} \end{bmatrix}. \tag{A9}$$

Using the chain rule, each entry of the matrix can be calculated as follows:

$$\frac{\partial f(Ax)}{\partial a_{ij}} = \frac{\partial f(y_1, y_2, \dots, y_m)}{\partial a_{ij}} = \frac{\partial f}{\partial y_1} \frac{\partial y_1}{\partial a_{ij}} + \frac{\partial f}{\partial y_2} \frac{\partial y_2}{\partial a_{ij}} + \dots + \frac{\partial f}{\partial y_i} \frac{\partial y_i}{\partial a_{ij}} = \frac{\partial f}{\partial y_i} x_j \tag{A10}$$

and hence (A9) can be rewritten as

$$\begin{bmatrix} \frac{\partial f}{\partial y_1} x_1 & \frac{\partial f}{\partial y_1} x_2 & \dots & \frac{\partial f}{\partial y_1} x_n \\ \frac{\partial f}{\partial y_2} x_1 & \frac{\partial f}{\partial y_2} x_2 & \dots & \frac{\partial f}{\partial y_2} x_n \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial y_m} x_1 & \frac{\partial f}{\partial y_m} x_2 & \dots & \frac{\partial f}{\partial y_m} x_n \end{bmatrix} = \frac{\partial f}{\partial y} x^T \tag{A11}$$

where  $\frac{\partial f}{\partial y}$  and  $x^T$  are  $m \times 1$  and  $1 \times n$  vectors respectively.

## References

1. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [[CrossRef](#)]
2. Guo, Y.; Liu, Y.; Oerlemans, A.; Lao, S.; Wu, S.; Lew, M.S. Deep learning for visual understanding: A review. *Neurocomputing* **2016**, *187*, 27–48. [[CrossRef](#)]
3. Yamashita, R.; Nishio, M.; Do, R.K.G.; Togashi, K. Convolutional neural networks: An overview and application in rology. *Insight Imaging* **2018**, *9*, 611–629. [[CrossRef](#)] [[PubMed](#)]
4. Fawaz, H.I.; Forestier, G.; Weber, J.; Idoumghar, L.; Muller, P.A. Deep learning for time series classification: A review. *Data Min. Knowl. Discov.* **2019**, *33*, 917–963. [[CrossRef](#)]
5. Zhao, Z.Q.; Zheng, P.; Xu, S.T.; Wu, X. Object Detection with Deep Learning: A Review. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 3212–3232. [[CrossRef](#)] [[PubMed](#)]
6. 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](#)]
7. 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](#)]
8. Zuo, C.; Qian, J.; Feng, S.; Yin, W.; Li, Y.; Fan, P.; Han, J.; Qian, K.; Chen, Q. Deep learning in optical metrology: A review. *Light Sci. Appl.* **2022**, *11*, 39. [[CrossRef](#)]
9. Mohammed, A.; Kora, R. A comprehensive review on ensemble deep learning: Opportunities and challenges. *J. King Saud Univ.-Comput. Inf. Sci.* **2023**, *35*, 757–774. [[CrossRef](#)]
10. Sharifani, K.; Amini, M. Machine Learning and Deep Learning: A Review of Methods and Applications. *World Inf. Technol. Eng. J.* **2023**, *10*, 3897–3904.
11. Khoei, T.T.; Slimane, H.O.; Kaabouch, N. Deep learning: Systematic review, models, challenges, and research directions. *Neural Comput. Appl.* **2023**, *35*, 23103–23124. [[CrossRef](#)]
12. Sarker, I.H. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *SN Comput. Sci.* **2021**, *2*, 420. [[CrossRef](#)] [[PubMed](#)]
13. Mankitsa, N.; Maraslidi, G.; Moysis, L.; Fragulis, G. A Review of Machine Learning and Deep Learning for Object Detection, Semantic Segmentation, and Human Action Recognition in Machine and Robotic Vision. *Technologies* **2024**, *12*, 15. [[CrossRef](#)]
14. Hazmoune, S.; Bougamouza, F. Using transformers for multimodal emotion recognition: Taxonomies and state of the art review. *Eng. Appl. Artif. Intell.* **2024**, *2*, 108339. [[CrossRef](#)]
15. Rudregowda, S.; Patilkulkarni, S.; Ravi, V.; Gururaj, H.L.; Krichen, M. Audiovisual speech recognition based on a deep convolutional neural network. *Data Sci. Manag.* **2024**, *7*, 25–34. [[CrossRef](#)]
16. Mehrish, A.; Majumder, N.; Bharadwaj, R.; Mihalcea, R.; Poria, S. A review of deep learning techniques for speech processing. *Inf. Fusion* **2023**, *99*, 101869. [[CrossRef](#)]
17. Jia, N.; Zheng, C. Emotion Speech Synthesis Method Based on Multi-Channel Time-Frequency Domain Generative Adversarial Networks (MC-TFD GANs) and Mixup. *Arab. J. Sci. Eng.* **2022**, *47*, 1749–1762. [[CrossRef](#)]
18. Alsobhani, A.; ALabboodi, H.M.A.; Mahdi, H. Speech Recognition using Convolution Deep Neural Networks. *J. Phys. Conf. Ser.* **2021**, *1973*, 012166. [[CrossRef](#)]
19. Liu, D.; Chen, L.; Wang, Z.; Diao, G. Speech Expression Multimodal Emotion Recognition Based on Deep Belief Network. *J. Grid Comput.* **2021**, *19*, 22. [[CrossRef](#)]
20. Amberkar, A.; Awasarmol, P.; Deshmukh, G.; Dave, P. Speech Recognition using Recurrent Neural Networks. In Proceedings of the 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT), Coimbatore, Tamil Nadu, India, 1–3 March 2018; pp. 1–4. [[CrossRef](#)]
21. Graves, A.; Mohamed, A.r.; Hinton, G. Speech recognition with deep recurrent neural networks. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech, and Signal Processing, Vancouver, Canada, 26–31 May 2013; pp. 6645–6649. [[CrossRef](#)]
22. Cherku, R.; Hussain, K.; Kavati, I.; Reddy, A.M.; Reddy, K.S. Sentiment classification with modified RoBERTa and recurrent neural networks. *Multimed. Tools Appl.* **2024**, *83*, 29399–29417. [[CrossRef](#)]
23. Alemu, A.A.; Melese, M.D.; Salau, A.O. Towards audio-based identification of Ethio-Semitic languages using recurrent neural network. *Sci. Rep.* **2023**, *13*, 19346. [[CrossRef](#)] [[PubMed](#)]
24. Tunstall, L.; von Werra, L.; Wolf, T. *Natural Language Processing with Transformers*; O'Reilly Media Inc.: Sebastopol, CA, USA, 2022.
25. Wang, W.; Gang, J. Application of Convolutional Neural Network in Natural Language Processing. In Proceedings of the 2018 International Conference on Information Systems and Computer Aided Education (ICISCAE), Changchun, China, 6–8 July 2018; pp. 64–70. [[CrossRef](#)]
26. Fathi, E.; Shoja, B.M. Chapter 9—Deep Neural Networks for Natural Language Processing. In *Handbook of Statistics*; Elsevier: Amsterdam, The Netherlands, 2018; Volume 38, pp. 229–316. [[CrossRef](#)]
27. Hang, L. Deep learning for natural language processing: Advantages and challenges. *Natl. Sci. Rev.* **2018**, *5*, 24–26. [[CrossRef](#)]
28. Goyal, P.; Pandey, S.; Jain, K. *Deep Learning for Natural Language Processing*; Apress: Berkeley, CA, USA, 2018. [[CrossRef](#)]
29. Sarikay, R.; Deoras, G.E.H.A. Application of Deep Belief Networks for Natural Language Understanding. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2014**, *22*, 778–784. [[CrossRef](#)]

30. Karwasra, R.; Khanna, K.; Suchal, K.; Sharma, A.; Singh, S. Protein structure prediction with recurrent neural network and convolutional neural network: A case study. In *Deep Learning Applications in Translational Bioinformatics*; Raza, K., Barh, D., Singh, D., Ahmad, N., Eds.; Academic Press: Cambridge, MA, USA, 2024; Chapter 13, pp. 211–229. [[CrossRef](#)]
31. Zhang, Z.X.; Xuan, J.; Yao, C.; Gao, Q.; Wang, L.; Jin, X.; Li, S. A deep learning approach for orphan gene identification in moso bamboo (*Phyllostachys edulis*) based on the CNN and Transformer model. *BMC Bioinform.* **2022**, *23*, 162. [[CrossRef](#)] [[PubMed](#)]
32. Xing, C.; Tian-Hao, L.; Yan, Z.; Chun-Chun, W.; Chi-Chi, Z. Deep-belief network for predicting potential miRNA-disease associations. *Brief. Bioinform.* **2021**, *22*, bbaa186. [[CrossRef](#)]
33. Shuting, J.; Xiangxiang, Z.; Feng, X.; Wei, H.; Xiangrong, L. Application of deep learning methods in biological networks. *Brief. Bioinform.* **2020**, *22*, 1902–1917. [[CrossRef](#)]
34. Md-Nafiz, H.; Friedberg, I. Identifying antimicrobial peptides using word embedding with deep recurrent neural networks. *Bioinformatics* **2018**, *35*, 2009–2016. [[CrossRef](#)]
35. Li, Y.; Huang, C.; Ding, L.; Li, Z.; Pan, Y.; Gao, X. Deep learning in bioinformatics: Introduction, application, and perspective in the big data era. *Methods* **2019**, *166*, 4–21. [[CrossRef](#)]
36. Min, S.; Lee, B.; Yoon, S. Deep learning in bioinformatics. *Brief. Bioinform.* **2017**, *18*, 851–869. [[CrossRef](#)]
37. Zeng, H.; Edwards, M.D.; Liu, G.; Gifford, D.K. Convolutional neural network architectures for predicting DNA–protein binding. *Bioinformatics* **2016**, *32*, i121–i127. [[CrossRef](#)]
38. Anand, D.; Khalaf, O.; Abdulsahib, G.; Chandra, G. Identification of meningioma tumor using recurrent neural networks. *J. Auton. Intell.* **2024**, *7*, 1–7. [[CrossRef](#)]
39. Han, Q.; Qian, X.; Xu, H.; Wu, K.; Meng, L.; Qiu, Z.; Weng, T.; Zhou, B.; Gao, X. DM-CNN: Dynamic Multi-scale Convolutional Neural Network with uncertainty quantification for medical image classification. *Comput. Biol. Med.* **2024**, *168*, 107758. [[CrossRef](#)] [[PubMed](#)]
40. Pu, Q.; Xi, Z.; Yin, S.; Zhao, Z.; Zhao, L. Advantages of transformer and its application for medical image segmentation: A survey. *BioMed. Eng. OnLine* **2014**, *23*, 14. [[CrossRef](#)] [[PubMed](#)]
41. Zhang, Y.; Wang, J.; Gorriz, J.M.; Wang, S. Deep Learning and Vision Transformer for Medical Image Analysis. *J. Imaging* **2023**, *9*, 147. [[CrossRef](#)] [[PubMed](#)]
42. Habashi, A.G.; Azab, A.M.; Eldawlatly, S.; Aly, G.M. Generative adversarial networks in EEG analysis An overview. *J. Neuroeng. Rehabil.* **2023**, *20*, 40. [[CrossRef](#)] [[PubMed](#)]
43. Elameer, A.S.; Jaber, M.M.; Abd, S.K. Radiography image analysis using cat swarm optimized deep belief networks. *J. Intell. Syst.* **2022**, *31*, 40–54. [[CrossRef](#)]
44. Wang, J.; Zhu, H.; Wang, S.H.; Zhang, Y.D. A Review of Deep Learning on Medical Image Analysis. *Mob. Netw. Appl.* **2021**, *26*, 351–380. [[CrossRef](#)]
45. Yu, H.; Yang, L.T.; Zhang, Q.; Armstrong, D.; Deen, M.J. Convolutional neural networks for medical image analysis State-of-the-art, comparisons, improvement and perspectives. *Neurocomputing* **2021**, *444*, 92–110. [[CrossRef](#)]
46. Qin, C.; Schlemper, J.; Caballero, J.; Price, A.N.; Hajnal, J.V.; Rueckert, D. Convolutional Recurrent Neural Networks for Dynamic MR Image Reconstruction. *IEEE Trans. Med. Imaging* **2019**, *38*, 280–290. [[CrossRef](#)]
47. Ker, J.; Wang, L.; Rao, J.; Lim, T. Deep Learning Applications in Medical Image Analysis. *IEEE Trans. Med. Imaging* **2018**, *6*, 9375–9389. [[CrossRef](#)]
48. Balmaceda-Huarte, R.; Baño-Medina, J.; Olmo, M.E.; Bettolli, M.L. On the use of convolutional neural networks for downscaling daily temperatures over southern South America in a climate change scenario. *Clim. Dyn.* **2024**, *62*, 383–397. [[CrossRef](#)]
49. Sheikh, Z.; Nia, A.M.; Ganjali, M. Climate change and anthropogenic effects on the drying of a saline lake in an arid region (Namak Lake, Iran). *Theor. Appl. Climatol.* **2024**, *155*, 715–734. [[CrossRef](#)]
50. Xu, G.; Wang, H.; Ji, S.; Ma, Y.; Feng, Y. MPformer: A transformer-based model for earthen ruins climate prediction. In *Tsinghua Science and Technology*; TUP: Beijing, China, 2024; pp. 1–10. [[CrossRef](#)]
51. Raj, S.; Tripathi, S.; Tripathi, K.C.; Bharti, R.K. Hybrid optimized deep recurrent neural network for atmospheric and oceanic parameters prediction by feature fusion and data augmentation model. *J. Comb. Optim.* **2024**, *47*, 66. [[CrossRef](#)]
52. Marano, G.C.; Rosso, M.M.; Aloisio, A.; Cirrincione, G. Generative adversarial networks review in earthquake-related engineering fields. *Bull. Earthq. Eng.* **2023**, *22*, 3511–3562. [[CrossRef](#)]
53. Wang, H.; Khayatnezhad, M.; Youssefi, N. Using an optimized soil and water assessment tool by deep belief networks to evaluate the impact of land use and climate change on water resources. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e6807. [[CrossRef](#)]
54. Alerskans, E.; Nyborg, J.; Birk, M.; Kaas, E. A transformer neural network for predicting near-surface temperature. *Meteorol. Appl.* **2022**, *29*, e298. [[CrossRef](#)]
55. Gao, Y.; Ruan, Y. Interpretable deep learning model for building energy consumption prediction based on attention mechanism. *Energy Build.* **2021**, *252*, 111379. [[CrossRef](#)]
56. Chattopadhyay, A.; Hassanzadeh, P.; Pasha, S. Predicting clustered weather patterns: A test case for applications of convolutional neural networks to spatio-temporal climate data. *Sci. Rep.* **2020**, *10*, 1317. [[CrossRef](#)]
57. Pang, Z.; Niu, F.; O'Neill, Z. Solar radiation prediction using recurrent neural network and artificial neural network: A case study with comparisons. *Renew. Energy* **2020**, *156*, 279–289. [[CrossRef](#)]

58. Ardabili, S.; Mosavi, A.; Dehghani, M.; Várkonyi-Kóczy, A.R. Deep Learning and Machine Learning in Hydrological Processes Climate Change and Earth Systems a Systematic Review. In *Engineering for Sustainable Future*; Várkonyi-Kóczy, A.R., Ed.; Springer International Publishing: Cham, Switzerland, 2020; pp. 52–62. [[CrossRef](#)]
59. Wiecha, P. Deep learning for nano-photonics materials—The solution to everything!? *Curr. Opin. Solid State Mater. Sci.* **2024**, *28*, 101129. [[CrossRef](#)]
60. Kale, A.P.; Wahul, R.M.; Patange, A.D.; Soman, R.; Ostachowicz, W. Development of Deep Belief Network for Tool Faults Recognition. *Sensors* **2023**, *23*, 1872. [[CrossRef](#)] [[PubMed](#)]
61. Shang, H.; Sun, C.; Liu, J.; Chen, X.; Yan, R. Defect-aware transformer network for intelligent visual surface defect detection. *Adv. Eng. Inform.* **2023**, *55*, 101882. [[CrossRef](#)]
62. Rane, N. Transformers in Material Science: Roles, Challenges, and Future Scope. *Chall. Future Scope* **2023**, 1–21. [[CrossRef](#)]
63. Luleci, F.; Catbas, F.N.; Avci, O. Generative adversarial networks for labeled acceleration data augmentation for structural damage detection. *J. Civ. Struct. Health Monit.* **2023**, *13*, 181–198. [[CrossRef](#)]
64. Rautela, M.; Gopalakrishnan, S. Ultrasonic guided wave based structural damage detection and localization using model assisted convolutional and recurrent neural networks. *Exp. Syst. Appl.* **2021**, *167*, 114189. [[CrossRef](#)]
65. Wang, B.; Lei, Y.; Yan, T.; Li, N.; Guo, L. Recurrent convolutional neural network: A new framework for remaining useful life prediction of machinery. *Neurocomputin* **2020**, *379*, 117–129. [[CrossRef](#)]
66. Yang, J.; Li, S.; Wang, Z.; Dong, H.; Wang, J.; Tang, S. Using Deep Learning to Detect Defects in Manufacturing: A Comprehensive Survey and Current Challenges. *Materials* **2020**, *13*, 5755. [[CrossRef](#)]
67. Sun, J.; Steinecker, A.; Glocker, P. Application of Deep Belief Networks for Precision Mechanism Quality Inspection. In *Precision Assembly Technologies and Systems*; Ratchev, S., Ed.; Springer: Berlin/Heidelberg, Germany, 2014; pp. 87–93. [[CrossRef](#)]
68. Cordeiro, J.R.; Raimundo, A.; Postolache, O.; Sebastiao, P. Neural Architecture Search for 1D CNNs—Different Approaches Tests and Measurements. *Sensors* **2021**, *21*, 7990. [[CrossRef](#)]
69. Kiranyaz, S.; Avci, O.; Abdeljaber, O.; Ince, T.; Gabbouj, M.; Inman, D.J. 1D convolutional neural networks and applications: A survey. *Mech. Syst. Signal Process.* **2021**, *151*, 107398. [[CrossRef](#)]
70. Luo, R.; Popp, J.; Bocklitz, T. Deep Learning for Raman Spectroscopy: A Review. *Analytica* **2022**, *3*, 287–301. [[CrossRef](#)]
71. Petmezas, G.; Stefanopoulos, L.; Kilintzis, V.; Tzavelis, A.; Rogers, J.A.; Katsaggelos, A.K.; Maglaveras, N. State-of-the-Art Deep Learning Methods on Electrocardiogram Data: Systematic Review. *JMIR Med. Inform.* **2022**, *10*, e38454. [[CrossRef](#)] [[PubMed](#)]
72. Mozaffari, M.H.; Tay, L.L. Overfitting One-Dimensional convolutional neural networks for Raman spectra identification. *Spectrochim. Acta A* **2022**, *272*, 120961. [[CrossRef](#)] [[PubMed](#)]
73. De Silva, U.; Koike-Akino, T.; Ma, R.; Yamashita, A.; Nakamizo, H. A Modular 1D-CNN Architecture for Real-time Digital Pre-distortion. In Proceedings of the 2022 IEEE Topical Conference on RF/Microwave Power Amplifiers for Radio and Wireless Applications (PAWR), Las Vegas, NV, USA, 16–19 January 2022; pp. 79–81. [[CrossRef](#)]
74. Mitiche, I.; Nesbitt, A.; Conner, S.; Boreham, P.; Morison, G. 1D-CNN based real-time fault detection system for power asset diagnostics. *IET Res. J.* **2020**, *14*, 5766–5773. [[CrossRef](#)]
75. Kiranyaz, S.; Zabihi, M.; Rad, A.B.; Ince, T.; Hamila, R.; Gabbouj, M. Real-time phonocardiogram anomaly detection by adaptive 1D Convolutional Neural Networks. *Neurocomputing* **2020**, *411*, 291–301. [[CrossRef](#)]
76. Huaxing, X.; Yunzhi, T.; Haichuan, R.; Xudong, L. A Lightweight Channel and Time Attention Enhanced 1D CNN Model for Environmental Sound Classification. *Exp. Syst. Appl.* **2024**, *249*, 123768. [[CrossRef](#)]
77. Allamy, S.; Koerich, A.L. 1D CNN Architectures for Music Genre Classification. In Proceedings of the 2021 IEEE Symposium Series on Computational Intelligence (SSCI), Orlando, FL, USA, 5–7 December 2021; pp. 1–7. [[CrossRef](#)]
78. Kim, T.; Lee, J.; Nam, J. Comparison and Analysis of SampleCNN Architectures for Audio Classification. *IEEE J. Sel. Top. Signal Process.* **2019**, *13*, 285–297. [[CrossRef](#)]
79. Hoshen, Y.; Weiss, R.J.; Wilson, K.W. Speech acoustic modeling from raw multichannel waveforms. In Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brisbane, QLD, Australia, 19–24 April 2015; pp. 4624–4628. [[CrossRef](#)]
80. Jain, M.; Bhardwaj, H.; Srivastav, A. Bayesian-Enhanced EEG Signal Analysis for Epileptic Seizure Recognition: A 1D-CNN LSTM Approach. In Proceedings of the 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 14–15 March 2024. [[CrossRef](#)]
81. Xu, G.; Ren, T.; Chen, Y.; Che, W. A One-Dimensional CNN-LSTM Model for Epileptic Seizure Recognition Using EEG Signal Analysis. *Front. Neurosci.* **2020**, *14*, 578126. [[CrossRef](#)]
82. Mattioli, F.; Porcaro, C.; Baldassarre, G. A 1D CNN for high accuracy classification and transfer learning in motor imagery EEG-based brain-computer interface. *J. Neural Eng.* **2022**, *18*, 066053. [[CrossRef](#)]
83. Zhao, D.; Jiang, R.; Feng, M.; Yang, J.; Wang, Y.; Hou, X.; Wang, X. A deep learning algorithm based on 1D CNN-LSTM for automatic sleep staging. *Technol. Health Care* **2022**, *30*, 323–336. [[CrossRef](#)]
84. Dutta, M.D.A. ECG Disease Classification Using 1D CNN. In Proceedings of the 2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI), Gwalior, India, 14–16 March 2024. [[CrossRef](#)]
85. Jasvitha, B.D.; Kanagaraj, K.; Murali, K.; Singh, T.; Duraisamy, P. 1D CNN Framework on ECG Signals. In Proceedings of the 2024 3rd International Conference for Innovation in Technology (INOCON), Bangalore, India, 1–3 March 2024; pp. 1–6. [[CrossRef](#)]

86. Li, F.; Liu, M.; Zhao, Y.; Kong, L.; Dong, L.; Liu, X.; Hui, M. Feature extraction and classification of heart sound using 1D convolutional neural networks. *EURASIP J. Adv. Signal Process.* **2019**, *59*, 59. [[CrossRef](#)]
87. Zhang, W.; Yu, L.; Ye, L.; Zhuang, W.; Ma, F. ECG Signal Classification with Deep Learning for Heart Disease Identification. In Proceedings of the 2018 International Conference on Big Data and Artificial Intelligence (BDAI), Beijing, China, 22–24 June 2018; pp. 47–51. [[CrossRef](#)]
88. Mian, T.S. An Unsupervised Neural Network Feature Selection and 1D Convolution Neural Network Classification for Screening of Parkinsonism. *Diagnostics* **2022**, *12*, 1796. [[CrossRef](#)] [[PubMed](#)]
89. Lella, K.K.; Pja, A. Automatic COVID-19 disease diagnosis using 1D convolutional neural network and augmentation with human respiratory sound based on parameters: Cough, breath, and voice. *AIMS Public Health* **2021**, *8*, 240–264. [[CrossRef](#)] [[PubMed](#)]
90. Cao, Y.; Näslund, I.; Näslund, E.; Ottosson, J.; Montgomery, S.; Stenberg, E. Using a Convolutional Neural Network to Predict Remission of Diabetes After Gastric Bypass Surgery. *JMIR Med. Inform.* **2021**, *9*, e25612. [[CrossRef](#)] [[PubMed](#)]
91. Saheed, Y.K.; Abdulganiyu, O.H.; Majikumna, K.U.; Mustapha, M.; Workneh, A.D. ResNet50-1D-CNN: A new lightweight resNet50-One-dimensional convolution neural network transfer learning-based approach for improved intrusion detection in cyber-physical systems. *Int. J. Crit. Infrastruct. Prot.* **2024**, *45*, 100674. [[CrossRef](#)]
92. Qazi, E.U.H.; Almorjan, A.; Zia, T. A One-Dimensional Convolutional Neural Network (1D-CNN) Based Deep Learning System for Network Intrusion Detection. *Appl. Sci.* **2022**, *12*, 7986. [[CrossRef](#)]
93. Meliboev, A.; Alikhanov, J.; Kim, W. Performance Evaluation of Deep Learning Based Network Intrusion Detection System across Multiple Balanced and Imbalanced Datasets. *Electronics* **2022**, *11*, 515. [[CrossRef](#)]
94. Huo, D.; Li, X.; Li, L.; Gao, Y.; Li, X.; Yuan, J. The Application of 1D-CNN in Microsoft Malware Detection. In Proceedings of the 2022 7th International Conference on Big Data Analytics (ICBDA), Guangzhou, China, 4–6 March 2022; pp. 181–187. [[CrossRef](#)]
95. Lam, N.T. Detecting Unauthorized Network Intrusion based on Network Traffic using Behavior Analysis Techniques. *Int. J. Adv. Comput. Sci. And Appl.* **2021**, *11*, 46–51. [[CrossRef](#)]
96. Akhtar, M.S.; Feng, T. Deep Learning-Based Framework for the Detection of Cyberattack Using Feature Engineering. *Secur. Commun. Netw.* **2021**, *2021*, 6129210. [[CrossRef](#)]
97. Gamal, M.; Abbas, H.M.; Moustafa, N.; Sitnikova, E.; Sadek, R.A. Few-Shot Learning for Discovering Anomalous Behaviors in Edge Networks. *Comput. Mater. Contin.* **2021**, *69*, 1823–1837. [[CrossRef](#)]
98. Meliboev, A.; Alikhanov, J.; Wooseong, K. 1D CNN based network intrusion detection with normalization on imbalanced data. In Proceedings of the 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIIC), Fukuoka, Japan, 19–21 February 2020; pp. 218–224. [[CrossRef](#)]
99. Liu, G.; Zhang, J. CNID: Research of Network Intrusion Detection Based on Convolutional Neural Network. *Discret. Dyn. Nat. Soc.* **2020**, *2020*, 4705982. [[CrossRef](#)]
100. Yang, H.; Wang, F. Wireless Network Intrusion Detection Based on Improved Convolutional Neural Network. *IEEE Access* **2019**, *7*, 64366–64374. [[CrossRef](#)]
101. Khan, R.U.; Zhang, X.; Alazab, M.; Kumar, R. An Improved Convolutional Neural Network Model for Intrusion Detection in Networks. In Proceedings of the 2019 Cybersecurity and Cyberforensics Conference (CCC), Melbourne, Australia, 8–9 May 2019; pp. 74–77. [[CrossRef](#)]
102. Ding, Y.; Zhai, Y. Intrusion detection system for NSL-KDD dataset using convolutional neural networks. In Proceedings of the (CSAI'18), Shenzhen, China, 8–10 December 2018; pp. 81–85. [[CrossRef](#)]
103. Shao, X.; Kim, C.S. Unsupervised Domain Adaptive 1D-CNN for Fault Diagnosis of Bearing. *Sensors* **2022**, *22*, 4156. [[CrossRef](#)] [[PubMed](#)]
104. Xie, X.; Wang, B.; Wan, T.; Tang, W. Multivariate Abnormal Detection for Industrial Control Systems Using 1D CNN and GRU. *IEEE Access* **2020**, *8*, 88348–88359. [[CrossRef](#)]
105. Flower, T.M.L.; Jaya, T. A novel concatenated 1D-CNN model for speech emotion cognition. *Biomed. Signal Process. Control* **2024**, *93*, 106201. [[CrossRef](#)]
106. Kwon, M.S. 1d-cnn: Speech emotion recognition system using a stacked network with dilated cnn features. *Comput. Mater. Contin.* **2021**, *67*, 4039–4059. [[CrossRef](#)]
107. Li, Y.; Baidoo, C.; Cai, T.; Kusi, G.A. Speech Emotion Recognition Using 1D CNN with No Attention. In Proceedings of the 2019 23rd International Computer Science and Engineering Conference (ICSEC), Pjuket, Thailand, 30 October–1 November 2019; pp. 351–356. [[CrossRef](#)]
108. Liu, L.; Si, Y.W. 1D convolutional neural networks for chart pattern classification in financial time series. *J. Supercomput.* **2022**, *78*, 14191–14214. [[CrossRef](#)]
109. Walsh, J.; Neupane, A.; Li, M. Evaluation of 1D convolutional neural network in estimation of mango dry matter content. *Spectrochim. Acta A Mol. Biomol. Spectrosc.* **2024**, *311*, 124003. [[CrossRef](#)]
110. Zeng, F.; Peng, W.; Kang, G.; Feng, Z.; Yue, X. Spectral Data Classification By One-Dimensional Convolutional Neural Networks. In Proceedings of the 2021 IEEE International Performance, Computing, and Communications Conference (IPCCC), Austin, TX, USA, 29–31 October 2021; pp. 1–6. [[CrossRef](#)]
111. Xiao, D.; Chen, Y.; Li, D.D.U. One-Dimensional Deep Learning Architecture for Fast Fluorescence Lifetime Imaging. *IEEE J. Sel. Top. Quantum Electron.* **2021**, *27*, 7000210. [[CrossRef](#)]



112. Riese, F.M.; Keller, S. Soil Texture Classification with 1D Convolutional Neural Networks based On Hyperspectral Data. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.* **2019**, *IV-2/W5*, 615–621. [[CrossRef](#)]
113. Liu, L.; Ji, M.; Buchroithner, M. Transfer Learning for Soil Spectroscopy Based on Convolutional Neural Networks and Its Application in Soil Clay Content Mapping Using Hyperspectral Imagery. *Sensors* **2018**, *18*, 3169. [[CrossRef](#)]
114. Hu, W.; Huang, Y.; Wei, L.; Zhang, F.; Li, H. Deep convolutional neural networks for hyperspectral image classification. *J. Sens.* **2015**, *2015*, 258619. [[CrossRef](#)]
115. Ng, W.; Minasny, B.; Montazerolghaem, M.; Padarian, J.; Ferguson, R.; Bailey, S.; McBratney, A.B. Convolutional neural network for simultaneous prediction of several soil properties using visible/near-infrared, mid-infrared, and their combined spectra. *Geoderma* **2019**, *352*, 251–267. [[CrossRef](#)]
116. Li, B.; Yu, Z.; Ke, X. One-dimensional convolutional neural network for mapping mineral prospectivity: A case study in Changba ore concentration area, Gansu Province. *Ore Geol. Rev.* **2024**, *160*, 105573. [[CrossRef](#)]
117. Xiong, Q.; Kong, Q.; Xiong, H.; Liao, L.; Yuan, C. Physics-informed deep 1D CNN compiled in extended state space fusion for seismic response modeling. *Comput. Struct.* **2024**, *291*, 107215. [[CrossRef](#)]
118. Nakano, M.; Sugiyama, D. Discriminating seismic events using 1D and 2D CNNs: Applications to volcanic and tectonic datasets. *Earth Planets Space* **2022**, *74*, 134. [[CrossRef](#)]
119. Jiang, S.; Zavala, V.M. Convolutional neural nets in chemical engineering: Foundations, computations, and applications. *AIChE J.* **2021**, *67*, e17282. [[CrossRef](#)]
120. Chen, X.; Kopsaftopoulos, F.; Wu, Q.; Ren, H.; Chang, F.K. A Self-Adaptive 1D Convolutional Neural Network for Flight-State Identification. *Sensors* **2019**, *19*, 275. [[CrossRef](#)]
121. Abdeljaber, O.; Avci, O.; Kiranyaz, S.; Gabbouj, M.; Inman, D.J. Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks. *J. Sound Vib.* **2017**, *388*, 154–170. [[CrossRef](#)]
122. Kiranyaz, S.; Ince, T.; Gabbouj, M. Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks. *IEEE Trans. Biomed. Eng.* **2016**, *63*, 664–675. [[CrossRef](#)]
123. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
124. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; Volume 32, pp. 1–12.
125. Ramachandran, P.; Zoph, B.; Le, Q.V. Searching for Activation Functions. *arXiv* **2017**. [[CrossRef](#)]
126. Nwankpa, C.; Ijomah, W.; Gachagan, A.; Marshall, S. Activation Functions: Comparison of trends in Practice and Research for Deep Learning. *arXiv* **2018**. [[CrossRef](#)]
127. Xavier, G.; Yoshua, B. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Machine Learning Research, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010; Teh, Y.W., Titterton, M., Eds.; Volume 9, pp. 249–256.
128. Trottier, L.; Giguère, P.; Chaib-draa, B. Parametric Exponential Linear Unit for Deep Convolutional Neural Networks. In Proceedings of the 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, Mexico, 18–21 December 2016; pp. 207–214.
129. Clevert, D.; Unterthiner, T.; Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In Proceedings of the 4th International Conference on Learning Representations (ICLR 2016), San Juan, Puerto Rico, 2–4 May 2016; Bengio, Y., LeCun, Y., Eds.; pp. 1–16.
130. Minai, A.A.; Williams, R.D. On the derivatives of the sigmoid. *Neural Netw.* **1993**, *6*, 845–853. [[CrossRef](#)]
131. Szandała, T. Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. In *Bio-Inspired Neurocomputing*; Bhoi, A.K., Mallick, P.K., Li, C.M., Balas, V.E., Eds.; Springer: Singapore, 2021; pp. 203–224. [[CrossRef](#)]
132. LeCun, Y.A.; Bottou, L.; Orr, G.B.; Müller, K.R. Efficient BackProp. In *Neural Networks: Tricks of the Trade: Second Edition*; Montavon, G., Orr, G.B., Müller, K.R., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 9–48. [[CrossRef](#)]
133. Vijayaprabakaran, K.; Sathiyamurthy, K. Towards activation function search for long short-term model network: A differential evolution based approach. *J. King Saud Univ.—Comput. Inf. Sci.* **2022**, *34*, 2637–2650. [[CrossRef](#)]
134. Tan, T.G.; Teo, J.; Anthony, P. A comparative investigation of nonlinear activation functions in neural controllers for search-based game AI engineering. *Artif. Intell. Rev.* **2014**, *41*, 1–25. [[CrossRef](#)]
135. Wang, Q.; Ma, Y.; Zhao, K.; Tian, Y. A Comprehensive Survey of Loss Functions in Machine Learning. *Ann. Data Sci.* **2022**, *9*, 187–212. [[CrossRef](#)]
136. Nwankpa, C.E.; Ijomah, W.L.; Gachagan, A.; Marshall, S. Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. In Proceedings of the 2nd International Conference on Computation Sciences and Technologies, Jamshoro, Pakistan, 17–19 December 2020; pp. 124–133.
137. Khalid, M.; Baber, J.; Kasi, M.K.; Bakhtyar, M.; Devi, V.; Sheikh, N. Empirical Evaluation of Activation Functions in Deep Convolution Neural Network for Facial Expression Recognition. In Proceedings of the 2020 43rd International Conference on Telecommunications and Signal Processing (TSP), Milan, Italy, 7–9 July 2020; pp. 204–207. [[CrossRef](#)]
138. Maas, A.L. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In Proceedings of the Proceedings of the 30th International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; Volume 28.

139. Li, Z.; Liu, F.; Yang, W.; Peng, S.; Zhou, J. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *33*, 6999–7019. [[CrossRef](#)]
140. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 11–18 December 2015; pp. 1026–1034. [[CrossRef](#)]
141. Montesinos López, O.A.; Montesinos López, A.; Crossa, J. Fundamentals of Artificial Neural Networks and Deep Learning. In *Multivariate Statistical Machine Learning Methods for Genomic Prediction*; Springer International Publishing: Cham, Switzerland, 2022; pp. 379–425. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.