applied sciences

MDPI

Article

FPGA Implementation of a BPSK 1D-CNN Demodulator

Yan Liu [1], Yue Shen [1], Li Li [2,3] and Hai Wang [4,*]

1 Key Laboratory of Electronic Equipment Structure Design, Ministry of Education, Xidian University, Xi'an 710071, China; liuy@xidian.edu.cn (Y.L.); yueshen@stu.xidian.edu.cn (Y.S.)
2 School of Aeronautics, Northwestern Polytechnical University, Xi'an 710072, China; lili_504@126.com
3 China Academy of Space Technology (Xi'an), Xi'an 710100, China
4 School of Aerospace Science and Technology, Xidian University, Xi'an 710071, China
* Correspondence: wanghai@mail.xidian.edu.cn; Tel.: +86-029-8189-1034

Received: 11 February 2018; Accepted: 14 March 2018; Published: 15 March 2018

Abstract: In this paper, we propose a field programmable gate array (FPGA) implementation of a one-dimensional convolution neural network (1D-CNN) demodulator for binary phase shift keying (BPSK). The 1D-CNN demodulator includes two 1D-CNNs and a decision module. Discrete time series of BPSK signals are imported into the well-trained 1D-CNNs. The 1D-CNNs detect the phase shifts' moment and type, including phase shift from 0 to π and that from π to 0. The decision module combines results of the two 1D-CNNs and outputs the demodulated data. In order to improve the efficiency of resource utilization and operation speed of the FPGA circuit, a time-delay network for convolutional calculation and a structure for piecewise approximation for the activation function were designed. To enhance the performance of the 1D-CNN demodulator, universal and diversity training data considering five impact factors were generated specially. Experimental results under different channel conditions show that the proposed demodulator has good adaptability to frequency offset and short latency. The demodulation loss of the proposed demodulator can almost be kept within 2 dB.

Keywords: BPSK; digital demodulation; neural networks; FPGA

1. Introduction

In recent years, communication technology has made a spurt of progress. As an important part of the communication process, demodulators have been widely used in satellite communications [1–4], navigation [5,6], mobile communications [7–9], and underwater communications [10,11], and so forth. Among them, due to their excellent antinoise performance, BPSK demodulators have attracted wide attention [12–14].

To realize a BPSK demodulator, the cooperation of hardware platforms and algorithms is necessary. So far, the most frequently used hardware platforms include analog circuits using discrete components, application-specific integrated circuits (ASIC), and general-purpose programmable devices. Analog circuit platforms have been rarely used in recent years, due to the instability of the discrete components [15]. ASICs have a small volume and low power consumption, but the cost is much higher and the function cannot be modified once the chip tape-out has been completed [16]. Among the general-purpose programmable devices, FPGAs have the advantages of remarkable stability, repeatable programming, and high energy efficiency. In recent years, FPGA implementation in BPSK demodulation has become a popular trend [17].

The traditional algorithms of BPSK demodulation can be divided into two categories: coherent demodulation [18,19] and noncoherent demodulation [20]. The coherent demodulation regenerates a local carrier, which has the same frequency and phase as the modulated carrier by carrier

synchronization; then mixes the local carrier with the modulated signal; commits to down-conversion, low-pass filtering, and timing synchronization; and finally completes demodulation. The noncoherent demodulation adds the delayed input signal to the current input signal, and then a pulse sequence can be formed. By analysis of the sequence, demodulated data can be obtained. As there is no need for carrier synchronization, noncoherent demodulation algorithms are easy to realize, but their performance is much lower than that of coherent demodulation algorithms. Nowadays, coherent demodulation is the most widely used BPSK demodulation method. However, it also has shortcomings, as follows: (1) sensitivity to the frequency offset; large frequency offsets may cause the carrier synchronization to malfunction; (2) high complexity, which means more hardware resource consumption and energy consumption; (3) the carrier synchronization needs too much time, which causes delay at the beginning of the communication task.

Recently, some scholars have attempted to introduce machine learning into demodulation technology. For the neural network demodulators mentioned in [21–23], the demodulation principle is to analyse the modulated data in every symbol period by neural network. The modulated data is divided into symbols according to the number of samplings. This may result in mistakes when offset exists. Furthermore, the design of the training data sets is not reasonable, as more nonideal factors need to be considered.

The convolution neural network (CNN) is an outstanding machine learning pattern [24]. Therein, due to the feature extraction ability and one-dimensional structure of the 1D-CNN [25], it is suitable for the treatment of discrete time series. It is noticed that in the published studies, there is almost no use of 1D-CNNs in demodulator research. The 1D-CNN is able to handle the problem of frequency offset if it is trained by appropriate training data, and the complexity can be simplified by a suitable network structure. By making full use of the parallel computing architecture of FPGA, the delay of demodulation may be reduced. Hence, this paper proposes the design and implementation of a BPSK demodulator based on 1D-CNN. In this paper, five nonideal factors were considered in the training data sets, so that the demodulator had a better adaptability to offsets. Experiments were repeatedly carried out to obtain a suitable network structure, considering both accuracy and complexity. A time-delay network for convolutional calculation and a structure for piecewise approximation of the activation function were designed, to further simplify calculation and shorten latency.

The rest of this paper is organized as follows: Section 2 introduces the basic principle of 1D-CNN demodulation. Section 3 describes the FPGA implementation of a 1D-CNN demodulator. In Section 4, the training and test process of the 1D-CNN demodulator are introduced, and their results are analyzed. Finally, Section 5 summarizes this paper.

2. Basic Principle of 1D-CNN Demodulation

The modulated BPSK signal in the time domain can be expressed as

$$s(t) = [\Sigma_n a_n g(t - nT_s)] \cos(\omega_c t + \varphi), \quad (1)$$

where $a_n = \pm A$, $g(t)$ denotes a single rectangular pulse, T_S is the pulse width, and ω_c is the angular frequency of the carrier. Information carried by the BPSK signal is contained in the phase shifts. Through the detection of the phase shifts, the carried information can be recovered.

The process of 1D-CNN demodulation is shown in Figure 1. The BPSK signal is imported separately into two 1D-CNNs. The 1D-CNN1 detects phase shift from 0 to π, and the 1D-CNN2 detects phase shift from π to 0. Each 1D-CNN outputs a pulse when a certain type of phase shift is met. A decision module is employed to handle the outputs of the 1D-CNNs. When the output of 1D-CNN1 is greater than the predefined threshold, the output of the decision module is converted from 1 to 0. When the output of 1D-CNN2 is greater than the predefined threshold, the output of the decision module is converted from 0 to 1.

The two 1D-CNNs have the same structure, only the parameters of the convolution kernels and neurons are different. The structure of the 1D-CNN is shown in Figure 2, which consists of four layers: an input layer, convolution layer, hidden layer, and output layer. The input layer conveys segmented data as an input vector. The convolution layer convolutes the input vector with a convolution kernel, and the result is transported to the hidden layer. The hidden layer aims to avoid the network being trapped in the local optimum, and to make 1D-CNN convergence easier during the training process. Neurons in the hidden layer are connected to the convolution layer. The weighted results from every neuron in the hidden layer are summed in the output layer, and then imported to the decision module. The operation process of 1D-CNNs includes forward propagation and backward propagation. In the training process, the backward propagation adjusts network parameters according to the output and loss, until the loss reaches the minimum. Once the network parameters are determined, the forward propagation can deduce the result independently.

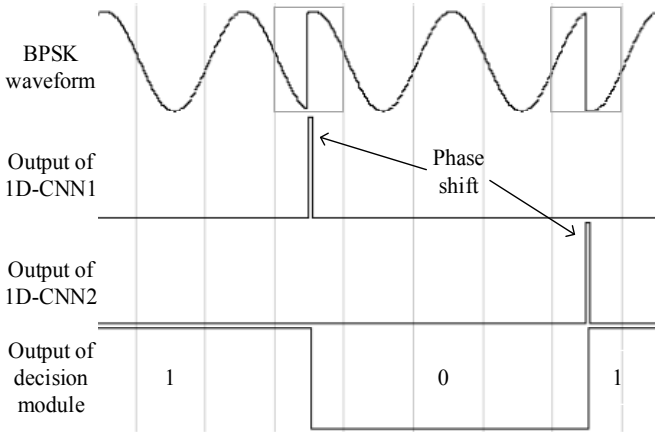

Figure 1. The process of one-dimensional convolution neural network (1D-CNN) demodulation. BPSK: binary phase shift keying.

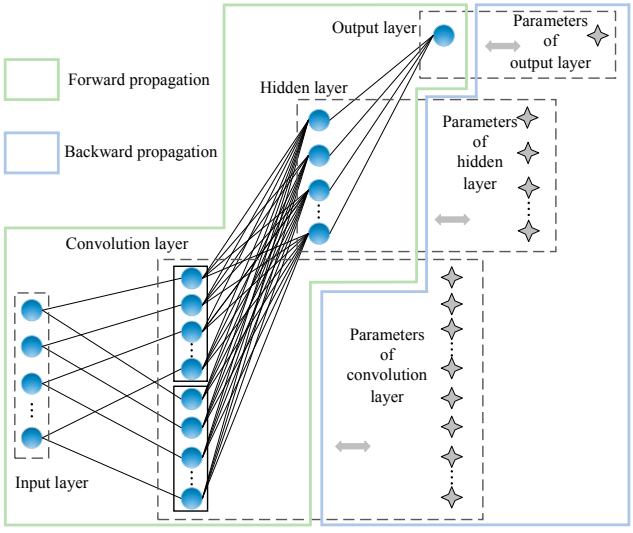

Figure 2. The structure of a 1D-CNN.

3. FPGA Implementation

This paper implements the 1D-CNN demodulator in an FPGA. The implementation block diagram is shown in Figure 3.

Two 1D-CNNs are implemented in an FPGA. The input of the 1D-CNN is a discrete time series from an analog-to-digital converter (ADC). The input layer segments the input vector by a sliding window, and conveys the input vector to the convolution layer. If we suppose that the input series is $x_1\ x_2 \ldots\ x_n$, where σ is the length of the input vector, then the input vector X_m can be indicated as

$$X_m = \begin{bmatrix} x(m) \\ x(m+1) \\ x(m+2) \\ \vdots \\ x(m+\sigma) \end{bmatrix}, \ m = 1, 2, 3 \ldots, \tag{2}$$

Registers are cascaded to realize a sliding window. The convolution layer includes two convolution kernels, which store the convolution parameters in block read-only memory (ROM) and complete convolution calculations. A rectified linear unit (ReLU) function is selected as the activation function of the convolution layer. The number of neurons in the hidden layer is set to 20, and the activation function of the hidden layer is a sigmoid function. A single neuron is set in the output layer. The existence of a certain type of phase shift can be concluded directly by referring to the state of this neuron. Comparators are used in the decision module to compare the predefined threshold with the results of the two 1D-CNNs. The valid bus gives the enable signal according to the states of the ADC and each layer. All components are driven by the same clock, provided by the external crystal.

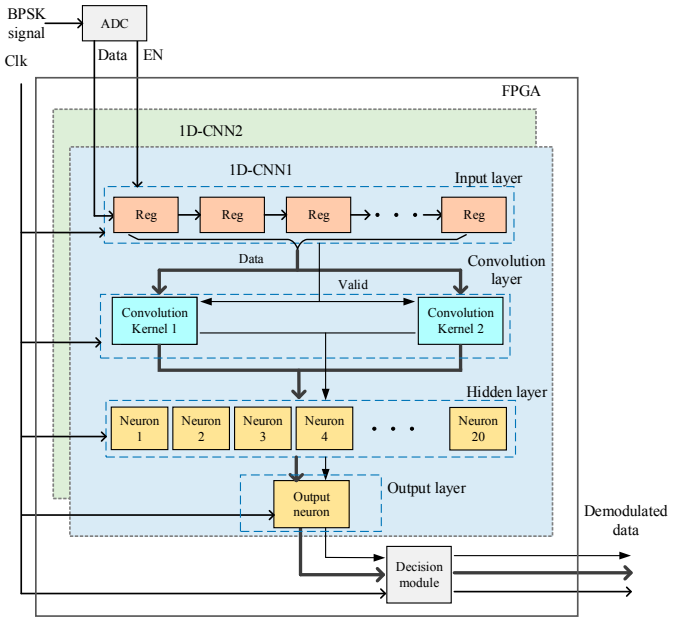

Figure 3. The FPGA (field programmable gate array) implementation block diagram. Reg: register; Clk: clock; ADC: analog-to-digital converter; EN: enable.

To improve the efficiency of resource utilization and operation speed of the FPGA, three methods were adopted as follows: (1) the use of a time-delay network for convolutional calculation; (2) the use of a look-up table (LUT) together with a piecewise function to achieve the activation function; (3) the use of a parallel structure within layers and a pipeline structure between layers.

3.1. Implementation of the Convolution Kernel

The convolution kernel is the core component of the convolutional layer. Repeated experiments show that the accuracy of detection reached the highest when the length of the input vector was

slightly larger than the sampling times of each carrier period. In the design, the length of the input vector is $M + 1$; M denoting the sampling times of each carrier period. In order to accurately detect the phase shifts in the input vectors, the convolution operation mode was always selected, meaning that the input vector and the output vector had the same length. According to the rule of convolution calculation, the output vector with the length of $M + 1$ can be obtained only if the input vector is expanded to the length of $2M + 1$. We used the element 0 to expand the vector. The expanded input vector convoluted with the convolution kernel $M + 1$ in length, and the result is shown in Figure 4.

$$\begin{bmatrix} 0 & \cdots & 0 & x_1 & x_2 & \cdots & x_{M+1} & 0 & \cdots & 0 \end{bmatrix} * \sum_{i=1}^{M+1} w_i =$$

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_{M+1} \end{bmatrix} \begin{bmatrix} w_{\frac{M}{2}+1} & w_{\frac{M}{2}} & \cdots \\ \vdots & \vdots & \\ w_{M+1} & w_M & \\ 0 & w_{M+1} & \cdots \\ 0 & 0 & \\ \vdots & \vdots & \\ 0 & 0 & \cdots \end{bmatrix} \begin{matrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ \vdots \\ w_{M+1} \end{matrix} \begin{bmatrix} \cdots & 0 & 0 \\ & \vdots & \vdots \\ & 0 & 0 \\ \cdots & w_1 & 0 \\ & w_2 & w_1 \\ & \vdots & \vdots \\ \cdots & w_{\frac{M}{2}} & w_{\frac{M}{2}+1} \end{bmatrix}$$

Figure 4. The convolution calculation result for the input vector and convolution kernel.

As shown in Figure 4, in order to complete this convolution calculation, $(M + 1)^2$ multiplication operations must be done. Some of the operations include the multiplier '0', which can be ignored. Calculation results of data in the line frame had all appeared in previous steps or would appear in the forthcoming steps. By feeding some of the results into the time-delay network that was composed of several time-delay queues, a large number of repeated calculations could be avoided. The structure of the time-delay network is shown in Figure 5.

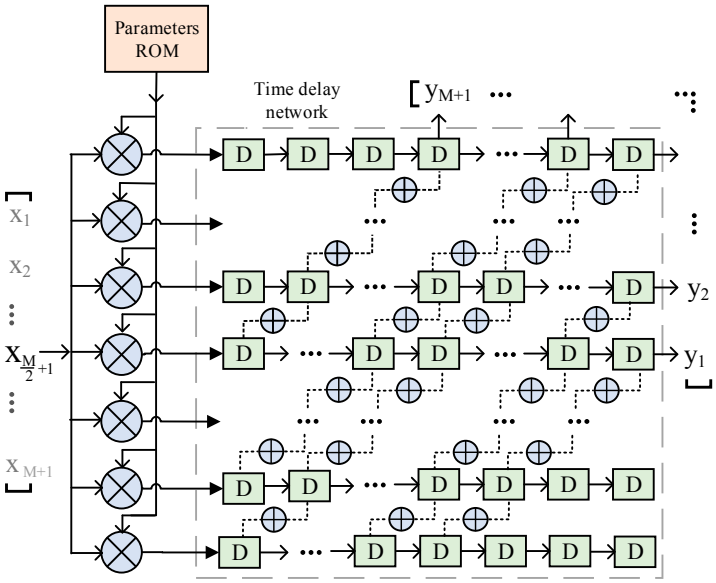

Figure 5. The structure of the time-delay network. ROM: read-only memory. D: delay unit; x: input; y: output; M: sampling times of each carrier period.

Assuming that $[x_1, x_2, \ldots, x_{M+1}]$ is the current input vector, together with data in the time-delay network, we can obtain the convolution result $[y_1, y_2, \ldots, y_{M+1}]$ by only $M + 1$ multiplication operations. The calculation complexity is reduced by one dimension. The cost of the time-delay queues is far less than the multipliers in FPGA implementation. Under the control of the clock in the FPGA, data can be accurately beat-delayed with the occupation of few hardware resources. The timing sequence of the time-delay network is shown in Figure 6.

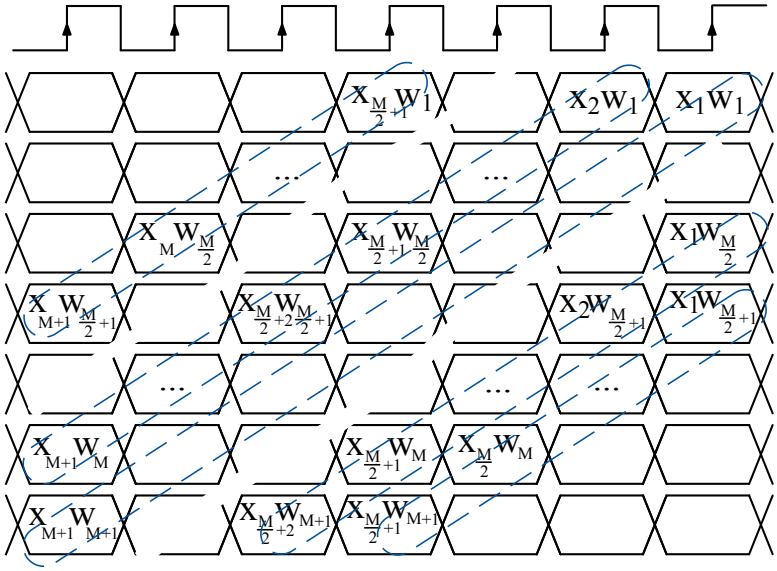

Figure 6. Timing sequence of the time delay network. X: input; W: convolution kernel.

3.2. Implementation of a Single Neuron

Neurons play an important role in the hidden layer. Assuming that $[p_1, p_2, \ldots, p_n]$ are the inputs of each neuron, $[W_1, W_2, \ldots, W_n]$ are the weights of the synapses, b is the bias, and $f(x)$ is the activation function, the structure of the neuron can be indicated as in Figure 7. Each neuron is fully connected to every convolution kernel output through synapses, and each synapse has its own weight. The weighted results are successively summed, biased, activated, and finally output. In FPGA implementation, a full adder circuit is used to achieve the summing operation. The fixed-point multiplication IP core is used for multiplication.

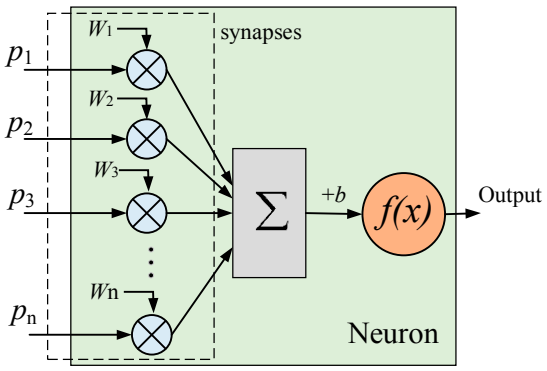

Figure 7. The structure of the neuron. P: neuron inputs; W: weights of synapses; b: bias; $f(x)$: activation function.

In this paper, two activation functions are used, which are:

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{3}$$

$$\text{ReLU}(x) = \max(0, x), \tag{4}$$

The implementation method of the ReLU function is very simple, so we mainly discuss the implementation method of the sigmoid function. Two methods are combined to realize the sigmoid function: the LUT method and the piecewise liner function method.

In the implementation of the LUT method, we mainly used the internal block RAM in the FPGA. First, we made a table from the input to the output, according to the resolution. Then, letting the input of the function be the address, the corresponding function value was written to the memory cell of this address. In this way, the complicated calculation process was simplified to the straightforward addressing process. By using the symmetry of the sigmoid function, only the positive part of the function was written to the LUT, whereas the negative part could be obtained by some simple adjustments according to the input data. This method has the advantages of high precision and ultrashort delay. However, the disadvantage is that the hardware resource cost is too high.

In the implementation of the piecewise liner function method, we used several linear functions for piecewise fitting of the sigmoid function. It is obvious that the more the function was segmented, the higher the accuracy, and the higher the resource consumption. This method is suitable for functions with good linearity. As for the sigmoid function, part of the interval can be replaced by linear functions, especially in sections near or far from the longitudinal axis.

In order to combine the advantages of the two methods, the LUT together with the piecewise function were adopted to achieve the sigmoid function. In three intervals with good linearity, which were $x \in (-1.3, 1.3) \cup (-7.5, -4) \cup (4, 7.5)$, three linear functions were used to replace the sigmoid function; in the intervals with bad linearity, i.e., $x \in (-4, -1.3) \cup (1.3, 4)$, the LUT was used to implement the sigmoid function; and in other intervals that were far away from the longitudinal axis, the sigmoid function converging to constants was replaced by constants. The structure of the sigmoid function implementation is shown in Figure 8. The control block firstly analyses the range of the input, then places the multiplexers (MUXs) at the right gear and adjusts the output value of the ROM. The output first-input/first-output (FIFO) is used to adjust the output delay of the LUT method. It aims to synchronize the two paths, and then synchronize the outputs of all neurons. This method takes into account both resource consumption and accuracy. The implementation result is shown in Figure 9.

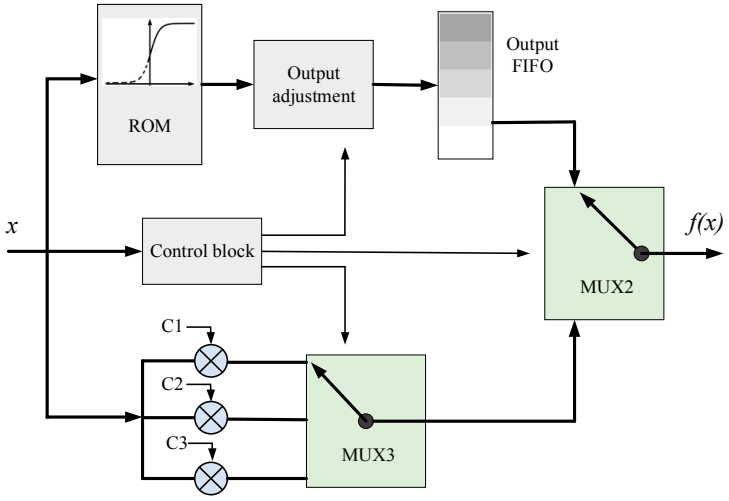

Figure 8. The implementation structure of the sigmoid function. C: constant; MUX: multiplexer.

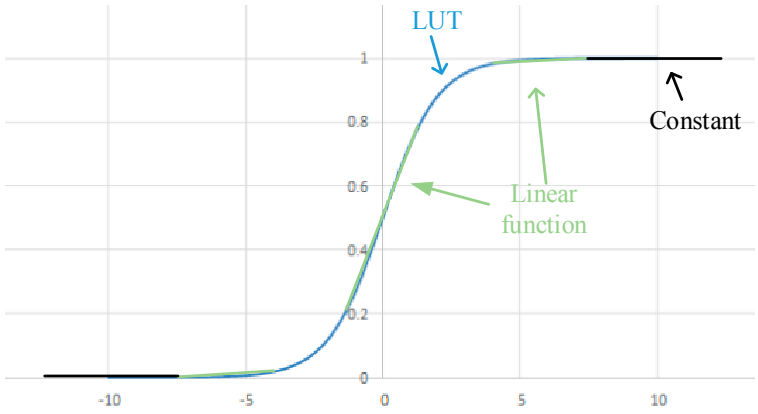

Figure 9. The implementation result of the sigmoid function.

3.3. Pipeline and Parallel Structure

The neural network runs in a pipelined mode between layers. The data are not cached from input to output. Layers are called every clock period. Each clock period, a set of data is fed into the 1D-CNN, and a set of results is output, which is delayed for several clock periods compared with the corresponding input. The pipelined mode maximizes the utilization of resources.

The pipelined mode between layers requires a parallel computing architecture inside each layer. The speed of the data stream is fixed in each layer, with no multiplexing of hardware resources between each step. In the convolution layer, two convolution kernels are routed respectively. Calculations of each element of the two output vectors are obtained synchronously; in the hidden layer, calculations of 20 parallel neurons are routed respectively. This parallel structure avoids the problem of routing across the clock domain in the implementation of the FPGA, and also avoids the problem of timing tension caused by frequent calls of the critical paths. The designed pipeline and parallel structure improves the stability of the circuit, and also provides the possibility to accelerate the speed of operations.

3.4. Precision and Quantization

In the training process of neural networks in a personal computer (PC), we chose the double precision floating-point type as the data type of neural network parameters, in order to obtain a network with high precision. In the FPGA implementation process, however, this kind of high-precision data type is neither feasible nor necessary. Taking this into account, the signed int type is chosen as the data type, meaning one bit for sign and 15 bits for data.

During the training process, the network input and parameters are normalized to unity; we treat the amplitude of the input signal as 1, and then obtain the network parameters of each layer. Here, the network parameters are represented as double precision floating-point, so we need to complete the data type conversion from floating-point to fixed-point. Differing from the floating-point numbers, fixed-point numbers have the problem of width expansion after multiplication. This means that the product of fixed-point numbers will become bigger and bigger, regardless of their actual value.

Assuming that f_1 and f_2 are two floating-point numbers, after fixed-point quantization with the coefficient n, their integer values are I_1 and I_2; therefore, $I_1 = [nf_1]$, $I_2 = [nf_2]$, and $I_1 I_2 = [n^2 f_1 f_2]$. However, the quantized actual value of the product of f_1 and f_2 is expected to be $R = [nf_1 f_2]$. It is clear that we can avoid the problem of width expansion through an additional division operation, that is:

$$R = \frac{I_1 I_2}{n}, \tag{5}$$

According to this characteristic, this paper chooses an integer power of 2 as the quantization coefficient, such that division operations can be replaced by bit shift operations. The actual product value can be expressed as

$$R = I_1 I_2 \gg (log_2 n), n = 2^k (k \in N), \tag{6}$$

Bit shift operation is very suitable for FPGA structure, occupying few hardware resources. Such a quantitative method greatly reduced the resource usage of multiplication. In the experiment, n was set to 2048. Examination of the results shows that no overflow happened during the operation process.

4. Experiment and Results

4.1. Experimental Platform

In this chapter, we describe the proposed 1D-CNN demodulator implementation using the Xilinx KC705 evaluation board. The block diagram of the experimental platform is shown in Figure 10. As shown in this diagram, a PC with a Keras neural network toolkit was used to generate the network parameters, which were later provided to a Xilinx KC705 evaluation board. The training data set X_i was provided in the neural network training process. In addition to the evaluation board itself, another three devices were applied to generate modulated data, which were the BPSK generator, FPGA mezzanine card (FMC) sampling subsystem, and additive white Gaussian noise (AWGN) generator.

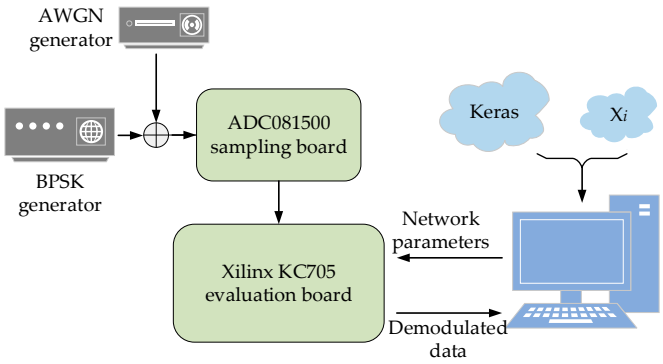

Figure 10. The block diagram of the experimental platform. X_i: training data set; AWGN: additive white Gaussian noise.

The experimental condition was set as follows: (1) carrier frequency was f_c = 10 MHz; (2) symbol rate was r_b = 5 Msps; (3) sampling frequency of the system was f_s = 80 MHz.

4.2. Training Data Sets

The demodulation performance of the 1D-CNN demodulator is obtained by training, so appropriate training data sets must be generated first. To enhance the adaptability of the 1D-CNN demodulator for a real channel condition, five nonideal factors were taken into consideration during the generation of training data X_i: signal-to-noise ratio (SNR), carrier frequency offset o_c, symbol rate offset o_b, sampling frequency offset o_s, and initial phase φ. Training data X_i can be regarded as the function of these five variables, which is

$$X_i = F(SNR, o_c, o_b, o_s, \varphi), \tag{7}$$

Empirically, we set the range of each variable, as shown in Table 1.

Table 1. The range of the variables.

Variables	Range	Variance Rate
SNR	0~15 (dB)	—
Carrier frequency offset	10 ± 0.1 (MHz)	10‰
Symbol rate offset	5 ± 0.01 (MHz)	2‰
Sampling frequency offset	80 ± 0.1 (MHz)	1.25‰
Initial phase	0~2π	—

MATLAB was employed to generate training data sets. Firstly, these five variables were assigned randomly in their respective range, and a set of values of the five variables was obtained. According to the carrier frequency, the symbol rate, and the initial phase, a BPSK-modulated waveform was generated, with the length fixed to 100 random symbols. Next, the noise signal of certain power was created referring to the SNR value, and the noise signal was added to the modulated waveform. Then, we extracted samples from the modulated waveform according to the sampling frequency. Finally, phase shifts labels were added to the sample sequence. In this way, the first data set X_1 was generated.

By repeating the above steps, a total of 1000 sets of training data were yielded. Next, the structures of the 1D-CNNs were built in a PC using a Keras neural network toolkit, and the generated training data sets X_1–X_{1000} were provided to train the neural network. Data sets X_1–X_{1000} were iterated for 100 times. The training loss curve is shown in Figure 11. It can be seen that the training process converged well. The well-trained network parameters were then imported to the FPGA.

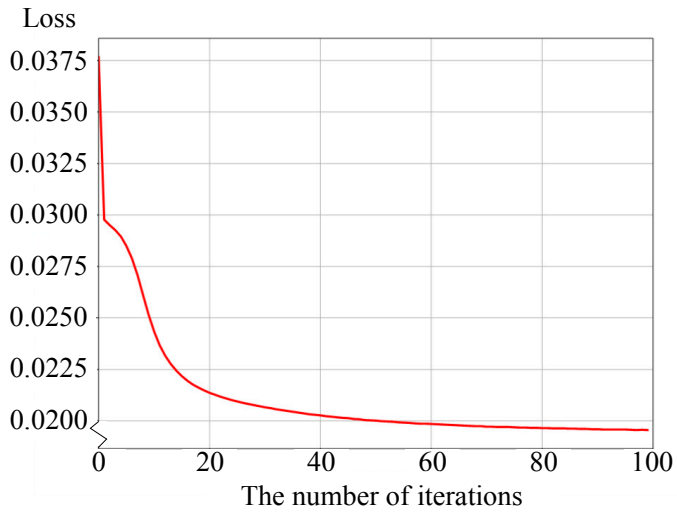

Figure 11. The curve of training loss.

4.3. Structure Parameters of the Network

In order to obtain the best network structure, several networks were designed by changing the two most important structure parameters: the length of the input vector and the number of neurons in the hidden layer. The modulated signals of different SNR were used to judge the performance of the network. As a representative case, the bit error rate (BER) results when SNR was 6 dB are shown in Figures 12 and 13.

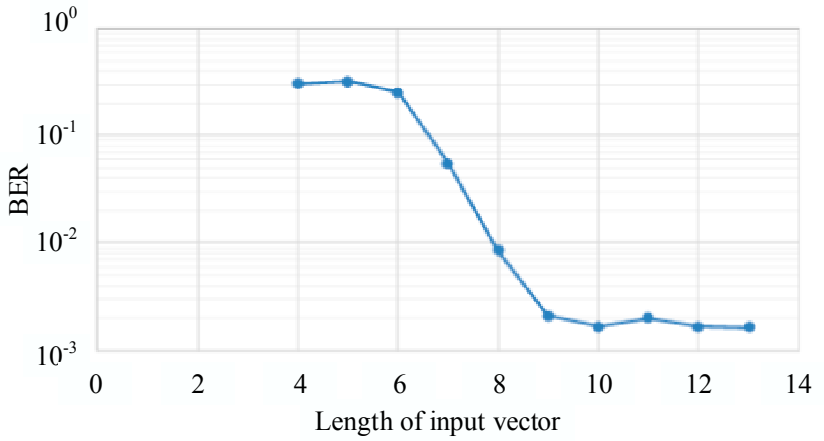

Figure 12. Performance under different length of input vector. BER: bit error rate.

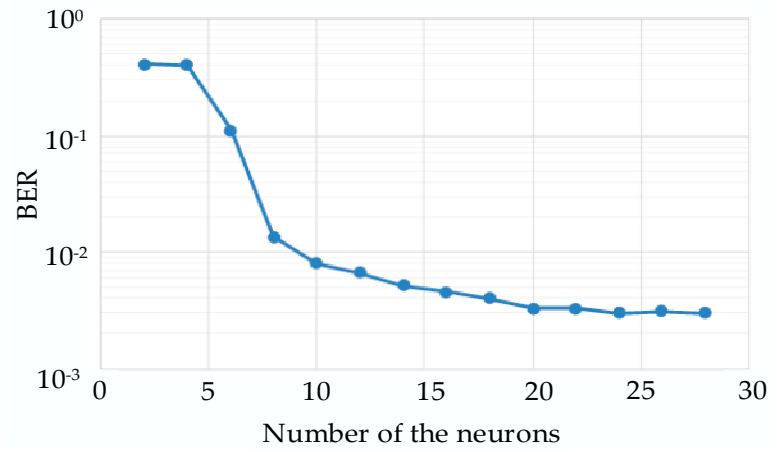

Figure 13. Performance under different numbers of neurons.

The result shown in Figure 12 indicates that the network cannot converge when the length of the input vector is too short (less than 4). The performance of the network was kept at an acceptable and stable level when the length of the input vector was slightly larger than the sampling number of each carrier period. In the experiment, the sampling number of each carrier period was 8, and the length of the input vector was set as 9.

The result shown in Figure 13 indicated that with the increase of the number of neurons in the hidden layer, the network performance gets better and better. However, change is not obvious when the number is more than 20. Considering that too many neurons may bring a greater amount of computation, the number was set as 20 in the experiment.

4.4. Results of the Implementation

The Xilinx KC705 evaluation board is equipped with a xc7k325t FPGA. The resource occupancy of the 1D-CNNs demodulator is shown in Table 2. We can see that the chosen FPGA can meet the requirements of the implementation. However, the occupancy rate of DSP48E1s is very high, so it was urgent to simplify the multiplication.

Table 2. Resource occupancy of the 1D-CNN demodulator.

Logic	Used	Available	Utilization
Slice registers	9552	407,600	2%
Slice LUTs	68,860	203,800	33%
Fully used LUT–FF pairs	2960	75,452	3%
Bonded IOBs	2	500	0%
Block RAM/FIFO	127	445	28%
BUFG/BUFGCTRLs	6	32	18%
DSP481Es	721	840	85%

In order to fully demonstrate the performance of the 1D-CNN demodulator, complexity, power consumption, and latency of the 1D-CNN demodulator and a of coherent demodulator were compared. In order to eliminate the influence of the hardware platform, the coherent demodulation algorithm was implemented in the same FPGA chip. A Costas loop and Gardner algorithm were employed in the coherent demodulation. The results are shown in Figure 14 and Table 3.

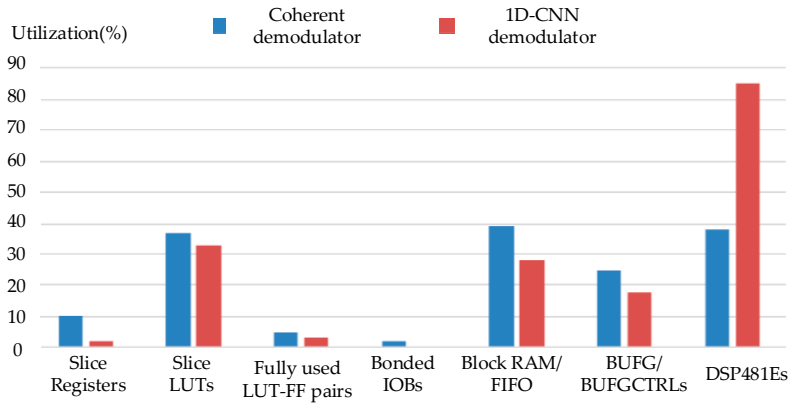

Figure 14. Resource utilization of the 1D-CNN demodulator and coherent demodulator.

Table 3. Power consumption and latency of the 1D-CNN demodulator and coherent demodulator.

	Coherent Demodulation	1D-CNN Demodulation	Saving
Power consumption (W)	0.584	0.531	9.07%
Time delay (ms)	0.12	0.004	96%

Hardware resource utilization is listed in Figure 14 to illustrate its complexity. The 1D-CNN demodulator consumes fewer slice registers, slice LUTs, block RAM, and more DSP481Es. This means that the 1D-CNN demodulator has simpler logic and a larger amount of calculation. Table 3 shows that the 1D-CNN demodulator saved power consumption by 9.07%. As for latency, because of the nonexistence of carrier synchronization, the 1D-CNN demodulator greatly shortened the delay, by 96%.

In order to illustrate the adaptability to frequency offset and symbol rate offset, the 1D-CNN demodulator and coherent demodulator were tested under the condition of the offset channel. In the offset channel, two of the abovementioned five factors, o_c and o_b, were set as 0.1 MHz and 5 kHz, respectively. The BER result is shown in Figure 15. It indicates that the 1D-CNN demodulator has the better adaptability to offset, especially in the case of low SNR.

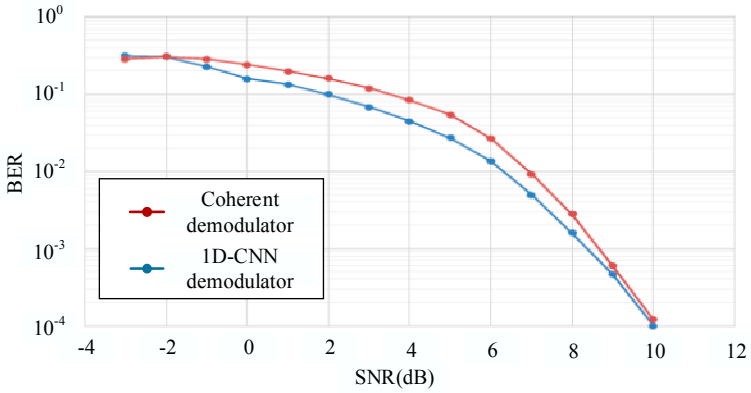

Figure 15. Performance of the 1D-CNN demodulator and coherent demodulator in the offset channel.

BER performance under an AWGN channel is generally regarded as an evaluation criterion for a demodulator. Theoretical value, simulation value, and actual tested value under the condition of an AWGN channel were carried out. The results are shown in Figure 16.

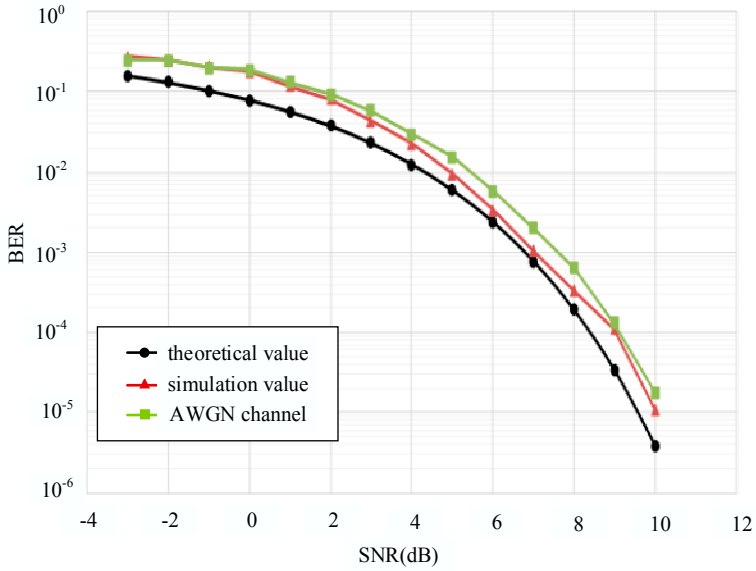

Figure 16. The test results in different channel conditions.

The experimental results show that under the condition of an AWGN channel, the demodulation loss of the 1D-CNN demodulator could be kept almost within 2 dB. This can be regarded as a relatively good performance, which can fully meet most of the requirements in wireless communication.

5. Conclusions

This paper presented an FPGA implementation of a 1D-CNN demodulator for BPSK. Two 1D-CNNs were contained in the 1D-CNN demodulator, to detect types and moments of the phase shift. A decision module was employed to synthesize results of the 1D-CNNs, and then the information carried by the modulation signal was obtained. A time-delay network for convolutional calculation and a structure for piecewise approximation for the activation function were adopted, improving the efficiency of resource utilization and operation speed. Universal and diversity training data were generated, strengthening the adaptability to the real channel condition. Complexity, power consumption, and latency of the 1D-CNN demodulator and a coherent demodulator were

compared. The result shows that the 1D-CNN demodulator had acceptable complexity, power consumption, and outstanding latency. The performance of the 1D-CNN demodulator was tested in different channels. In complicated channels where high offset was introduced, the 1D-CNN demodulator showed better adaptability to frequency offset. In an AWGN channel, the demodulation loss of the 1D-CNN demodulator could be kept almost within 2 dB. Owing to the good performance in the AWGN channel, the proposed 1D-CNN demodulator can meet most of the requirements in wireless communication. However, deficiencies of the 1D-CNN demodulator cannot be ignored; its complexity may be further reduced, and its adaptability to other nonideal factors like multipath effects should be considered.

Acknowledgments: This research is supported by National Natural Science Foundation of China (No. 51505358) and the Fundamental Research Funds for the Central Universities of China (No. JB160409).

Author Contributions: Hai Wang and Yan Liu conceived and designed the experiments; Li Li performed the experiments; Yan Liu and Yue Shen analyzed the data; Yan Liu contributed analysis tools; and Yue Shen and Hai Wang wrote the paper.

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

References

1. Han, L.L.; Seong-Mo, M.; Moon-Que, L.; Jong, W.Y. Six-port QPSK demodulator for optimal K-band multiport amplifier calibration. *Electron. Lett.* **2014**, *50*, 617–618.
2. Gunther, M.A.S.; Ricard, A.; Nick, J.; Roberto, M.; Enrico, V. GMSK Demodulator Implementation for ESA Deep-Space Missions. *Proc. IEEE* **2007**, *95*, 2132–2141.
3. Mehmet, R.Y.; Wentai, L.; John, D.; Bhaskar, B.; Paul, D.F.; Numan, S.D. SOI CMOS Implementation of a Multirate PSK Demodulator for Space Communications. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2007**, *54*, 420–431.
4. Timo, K.; Sascha, S.; Leonhard, R. A Wireless Sensor Network Using GNSS Receivers for a Short-Term Assessment of the Modal Properties of the Neckartal Bridge. *Appl. Sci.* **2017**, *7*, 626.
5. Jungbeom, K.; Junesol, S.; Heekwon, N.; Deokhwa, H.; Donguk, K.; Byungwoon, P.; Changdon, K. Accuracy Improvement of DGPS for Low-Cost Single-Frequency Receiver Using Modified Flächen Korrektur Parameter Correction. *ISPRS Int. J. Geo-Inf.* **2017**, *6*, 222.
6. Shi, Y.C.; Zhang, L.W.; Liu, Y.B. An embedded high sensitivity navigation receiver for GPS. In Proceedings of the 2011 International Conference on Consumer Electronics, Communications and Networks, Xianning, China, 16–18 April 2011; pp. 1023–1026.
7. Ro, J.-H.; Kim, J.-K.; You, Y.-H.; Song, H.-K. Linear Approximation Signal Detection Scheme in MIMO-OFDM Systems. *Appl. Sci.* **2018**, *8*, 49.
8. Kyungtae, K.; Junhee, R.; Dong, K.N. Accommodating the Variable Timing of Software AES Decryption on Mobile Receivers. *IEEE Syst. J.* **2014**, *8*, 726–736. [CrossRef]
9. Petrellis, N. Undersampling in Orthogonal Frequency Division Multiplexing Telecommunication Systems. *Appl. Sci.* **2014**, *4*, 79–98. [CrossRef]
10. Rami, A.; Heba, S.; Mohamad, A.E. A Novel Fractional Fourier Transform-Based ASK-OFDM System for Underwater Acoustic Communications. *Appl. Sci.* **2017**, *7*, 1286.
11. Baek, C.U.; Jung, J.W.; Do, D.W. Study on the Structure of an Efficient Receiver for Covert Underwater Communication Using Direct Sequence Spread Spectrum. *Appl. Sci.* **2018**, *8*, 58. [CrossRef]
12. Hyunwoo, C.; Hyungwoo, L.; Joonsung, B.; Hoi-Jun, Y. A 5.2 mW IEEE 802.15.6 HBC Standard Compatible Transceiver with Power Efficient Delay-Locked-Loop Based BPSK Demodulator. *IEEE J. Solid-State Circuits* **2015**, *50*, 2549–2559.
13. Yan, H.; Macias-Montero, J.G.; Akhnoukh, A.; de Vreede, L.C.N.; Long, J.R.; Burghartz, J.N. An Ultra-Low-Power BPSK Receiver and Demodulator Based on Injection-Locked Oscillators. *IEEE Trans. Microwav. Theory Tech.* **2011**, *59*, 1339–1349. [CrossRef]
14. Hiroyuki, T.; Toshihiko, K.; Akihiko, H.; Koichi, M.; Naoya, K. 10-Gbit/s BPSK Modulator and Demodulator for a 120-GHz-Band Wireless Link. *IEEE Trans. Microwav. Theory Tech.* **2011**, *59*, 1361–1368.

15. Wu, Z.H.; Liang, Z.M.; Li, B. A new BPSK demodulation circuit for command transmission in wireless implantable neural recording system. In Proceedings of the 2008 IEEE Asia Pacific Conference on Circuits and Systems, Macao, China, 30 November–3 December 2008; pp. 1526–1528.
16. Macias-Montero, J.G.; Yan, H.; Akhnoukh, A.; de Vreede, L.C.N.; Long, J.R.; Lopez-Villegas, J.M.; Pekarik, J.J. A 19GHz 250pJ/bit non-linear BPSK demodulator in 90nm CMOS. In Proceedings of the 2009 ESSCIRC, Athens, Greece, 14–18 September 2009; pp. 304–307.
17. Roshna, T.R.; Nivin, R.; Sherly, J.; Apren, T.J.; Alex, V. Design and Implementation of Digital Costas Loop and Bit Synchronizer in FPGA for BPSK Demodulation. In Proceedings of the 2013 International Conference on Control Communication and Computing (ICCC), Thiruvananthapuram, India, 13–15 December 2013; pp. 39–44.
18. Yamu, H.; Mohamad, S. A fully integrated low-power BPSK demodulator for implantable medical devices. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2005**, *52*, 2552–2562. [CrossRef]
19. Li, M.; Li, B.W.; Song, Y.J.; Zhang, X.P.; Chang, L.Q.; Liu, J. Investigation of Costas Loop Synchronization Effect on BER Performance of Space Uplink Optical Communication System With BPSK Scheme. *IEEE Photonics J.* **2015**, *7*, 1–9. [CrossRef]
20. Wilkerson, B.P.; Kang, J.K. A non-coherent BPSK receiver with dual band filtering for implantable biomedical devices. In Proceedings of the 19th IEEE International Conference on Electronics Circuits and Systems, Seville, Spain, 9–12 December 2012; pp. 697–700.
21. Mohammad, R.A.; Einollah, B. Improving ANN BFSK demodulator performance with training data sequence sent by transmitter. In Proceedings of the 2010 Second International Conference on Machine Learning and Computing, Bangalore, India, 9–11 February 2010; pp. 276–281.
22. He, F.M.; Xu, X.Z.; Zhou, L.; Man, H. A learning based cognitive radio receiver. In Proceedings of the MILCOM 2011 Military Communications Conference, Baltimore, MD, USA, 7–10 November 2011; pp. 7–12.
23. Önder, M.; Akan, A.; Doğan, H. Advanced neural network receiver design to combat multiple channel impairments. *Turk. J. Electr. Eng. Comput. Sci.* **2016**, *24*, 3066–3077. [CrossRef]
24. Wang, H.; Shao, M.J.; Liu, Y.; Zhao, W. Enhanced Efficiency 3D Convolution Based on Optimal FPGA Accelerator. *IEEE Access* **2017**, *5*, 6909–6916. [CrossRef]
25. Turker, I.; Serkan, K.; Levent, E.; Murat, A.; Moncef, G. Motor Fault Detection by 1D Convolutional Neural Networks. *IEEE Trans. Ind. Electron.* **2016**, *63*, 7067–7075.