

Studies in Big Data 146

Marcin Korytkowski

Advanced Techniques of Artificial Intelligence in IT Security Systems

 Springer

Studies in Big Data

Volume 146

Series Editor

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland

The series “Studies in Big Data” (SBD) publishes new developments and advances in the various areas of Big Data- quickly and with a high quality. The intent is to cover the theory, research, development, and applications of Big Data, as embedded in the fields of engineering, computer science, physics, economics and life sciences. The books of the series refer to the analysis and understanding of large, complex, and/or distributed data sets generated from recent digital sources coming from sensors or other physical instruments as well as simulations, crowd sourcing, social networks or other internet transactions, such as emails or video click streams and other. The series contains monographs, lecture notes and edited volumes in Big Data spanning the areas of computational intelligence including neural networks, evolutionary computation, soft computing, fuzzy systems, as well as artificial intelligence, data mining, modern statistics and Operations research, as well as self-organizing systems. Of particular value to both the contributors and the readership are the short publication timeframe and the world-wide distribution, which enable both wide and rapid dissemination of research output.

The books of this series are reviewed in a single blind peer review process.


Indexed by SCOPUS, EI Compendex, SCIMAGO and zbMATH.

All books published in the series are submitted for consideration in Web of Science.

Marcin Korytkowski

Advanced Techniques of Artificial Intelligence in IT Security Systems

 Springer

Marcin Korytkowski 
Department of Intelligent Computer
Systems
Częstochowa University of Technology
Częstochowa, Poland

ISSN 2197-6503
Studies in Big Data

ISSN 2197-6511 (electronic)

ISBN 978-3-031-53853-7

ISBN 978-3-031-53854-4 (eBook)

<https://doi.org/10.1007/978-3-031-53854-4>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2024

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Paper in this product is recyclable.

Contents

1	Introduction	1
2	Artificial Neural Networks	7
2.1	Initialization of Weights Values in Neural Networks	12
2.2	Convolution Neural Networks	13
3	Application of Artificial Intelligence Methods in Profiling	
	Computer Network Users	19
3.1	Profiling Users Using the Random Forest Algorithm	23
3.2	Convolutional Networks in User Profiling	27
3.3	User Profiling Using Recurrent-Convolutional Networks	30
3.4	Detection of Network Traffic Anomalies Using OC-NN	33
4	Convolutional Glial Networks	43
4.1	Optimization of Convolutional Networks Using Glial Cells	45
4.2	The Use of Glial Cells to Optimize the Structure of a Convolutional Network with a Known Structure on the Example of an Image Classification Task	46
4.3	The Use of Glial Cells to Optimize the Structure of the Convolutional Network When Solving Problems Using Previously Unknown Models	50
4.4	Using Glial Cells to Extract Knowledge from a CNN Network	51
4.5	Glial Cells in the Improvement of Network Security	58
4.6	Detection of Phishing Threats	63
5	Anomaly Detection Using Sequence Patterns	73
5.1	Neural Networks in Sequence Pattern Detection	76
5.2	Generating Sequences	76
5.3	U-Net Network in Sequence Pattern Recognition	78
5.4	Network Training for Detecting Hidden Sequences	80
6	Concluding Remarks and Challenges for Future Research	85
	Bibliography	87

Chapter 1

Introduction



The development of the Internet—without which it is hard to imagine today’s world—entails the need to ensure the safety of its users. This problem is of an interdisciplinary nature, as researchers in various fields of IT [1, 2], mathematics, psychology (behavioral analysis) [3–5] to medicine (telemedicine) are working on it. The importance of this problem has increased significantly with the spread of the use of Internet resources during the COVID-19 pandemic, when the most important aspects of our lives, in particular those related to payments, were transferred to the virtual world. At the same time, the dynamic development of technology brought new challenges for the creators of IT systems, unknown until two decades before, related to ensuring the security of data collected and stored in social networks, which can be used to build user profiles and influence the behavior of their owners based on this knowledge [1, 2]. Suffice it to say, that the company Cambridge Analytica, which boasts on its website that it can influence the results of elections. Another consequence of the COVID pandemic is that it has resulted in a huge increase in the number of non-advanced Internet users—persons with very little experience and knowledge of the risks involved and with a low level of digital competence. This group includes children and young persons studying, but also elderly persons, who had to switch to performing their work duties remotely from day to day. One of the biggest threats to data security is data theft. It is obvious that data leakage and systems compromising have a very negative impact on the functioning of governments, local governments, private companies, hospitals, etc. Such attacks most often exploit gaps in firewalls or unified threat management (UTM) tools. Unfortunately, detection of such software errors or misconfigurations is possible only after they occur. However, it is worth noting that the amount of information that is transmitted by devices active in networks is impossible to analyze by a human without appropriate tools. Such issues can be attributed to the class of problems from the family of Big Data.

Network traffic generated by users, recorded in the form of log files, creates huge data resources. Their analysis by a human (e.g. an IT security expert), without appropriate tools, is practically impossible. It is worth emphasizing that time is of

paramount importance in the process of analyzing network events. At the same time, if a network attack occurs, in order to detect it, it is often necessary to link events that occurred over long periods. The time of analysis of log data depends on the complexity of applied algorithms and the hardware on which they are run. Today, the problem of information security concerns every field in which computers are used. Large corporations, public institutions such as municipal authorities, schools, and small companies and households need to ensure the security of their system and network resources. The degree to which protection is ensured depends largely on the available financial resources—the more financial resources entities have, the greater are their choice options in terms of hiring experts and buying high-quality equipment.

As mentioned earlier, a significant factor in this process is the human and their knowledge and skills. Currently, there are no solutions on the market that would ensure the security of computer networks without the participation of a human. The natural approach, therefore, is to use computational intelligence techniques to analyze the collected information.

State-of-the-art network solutions use artificial intelligence algorithms—including neural networks, which have recently become especially popular—to support data analysis, for example, through grouping and visualization.

The quality of neural networks depends on a high degree of the available training data. In particular, it should be representative of a given problem and there should be a sufficient amount of it.

Thanks to the use of the latest technologies, the processing of Big Data files and machine learning can be successfully implemented in the security of IT systems. However, there are still several challenges associated with the use of neural networks in this field. The first of them comes down to choosing the structure of the neural network that we want to use to solve a given task. Another is to obtain data that accurately represents a specific issue, which will be used during training. An important problem is also the computational complexity of the training processes and the operation of already learned structures. It is worth noting that the use of convolutional networks to solve such complex problems became possible only a few years ago. It was influenced by the results of research on the possibility of using the computing power of GPUs (e.g. CUDA technology) in the processes of training and testing neural networks.

This book presents techniques for ensuring the security of IT systems described in the literature and proprietary solutions based on artificial intelligence methods. The topic is extremely broad, so only selected issues will be discussed, in particular on the areas of profiling users and systems of the network, preventing leaks of sensitive (personal) data, crucial for the functioning of entities, and defense against phishing attacks. Furthermore, innovative structures of so-called glial networks will be presented. They might be helpful when we attempt to interpret the knowledge stored in convolutional networks. It is worth emphasizing that all the proposed solutions can be used after a simple adaptation in various areas, not only those related to security. Their advantage over methods described in the current literature will be presented. Today, the problem of information security concerns every field in

which computers are used. Large corporations, public institutions such as municipal authorities, schools, and small companies and households need to ensure the security of their system and network resources. The degree of protection depends to a large degree on the available financial resources. As mentioned earlier, a significant factor in this process is the human and their knowledge and skills. Currently, there are no solutions on the market that would ensure the security of computer networks without the participation of a human. State-of-the-art network solutions use artificial intelligence algorithms to support data analysis, for example, through grouping and visualization.

In order to ensure network security, devices such as UTM or firewalls are most often used. The main task of the firewall is to block network connections if they do not meet the criteria predefined by the administrator, e.g. they refer to a wrong TCP port, transmission occurs from an unauthorized IP address or MAC address, or over unauthorized network protocols. Often more advanced devices are equipped with an intrusion prevention system [6]. Their task is to protect network users by filtering network packets according to established rules or using an IDS—intrusion detection system in network-based IDS (NIDS). These tools require constant supervision and optimization from the administrator since most changes in the network require the addition of new rules to threat detection systems.

Regardless of the strategy adopted, a firewall does not analyze the transmitted traffic for potential threats in the permitted rules. A more advanced tool are IPS. Their task is to block traffic based on reports from modules such as IDS. The task of IDS is to identify whether the observed network traffic is allowed.

Artificial intelligence is not widely used in existing network tools. Most hardware manufacturers have their own approach to using machine learning to build security policies, and so far none has been standardized. The most popular solutions are next-generation firewalls. Recognized companies offer solutions to automate network security management processes. An example of such solutions is XSIAM of the company Palo Alto Networks. Until now, the tools of this company analyzed logs from networks collected on the basis of SIEM. However, the architects of Palo Alto noticed that the response time of the system was too long, even several days. Therefore a state-of-the-art AI tool has been created. It gathers detailed data—not just log entries and alerts—to stimulate machine learning for natively autonomous responses, such as correlating alerts and data, detecting highly advanced threats, and automatic correction based on native threat analysis and attack surface data.

The second leading company that uses AI is Fortinet. Since December 2019, neural networks have been used both in the research laboratory for analyzing transmitted threats and in FortiEDR—a solution that analyzes the behavior of network users in real-time. Based on the received information, an appropriate scenario is executed. The solution primarily detects malware on users' devices. In the area of adaptive firewalls using AI, F5 can boast the biggest achievements with its BIG-IP ASM products. In the initial phase, the system identifies the correct use of the application and begins to build a security policy based on a statistical analysis of the traffic and expected behavior of the application. In this phase, the system does not block requests until the Policy Builder program finishes processing an appropriate number

of network traffic events and building of security policy elements. In the next phase, the Policy Builder improves and tightens the security policy over time, until the number of changes of the existing policy stabilizes. In the final phase, Policy Builder further refines the security policy, until it is ready to take over oversight of the security features. Example of the system operation: A security strategy is created using the automatic policy-building option. The client sends a GET request for a /sell.php file. As no PHP-type file exists yet in the rules, the BIG-IP ASM system allows for the request but marks PHP as an invalid file type and sends a copy of the request to the Policy Builder program. The system continues flagging requests for PHP files as illegal until the request threshold is reached for that file type. After reaching the threshold for PHP files, the BIG-IP ASM system updates the security policy and no longer considers PHP to be an invalid file type [7–9].

One security extension can be HIDS—host-based intrusion detection systems—which, unlike IDS, works only on one device and in the classic version enable detection if an intrusion occurred in the past. Their functioning is based on Edmond Locard’s statement that “every contact leaves traces” [10], which in criminology means that a criminal leaves something at the crime scene and takes something from there as well. These techniques involve in-depth analysis of the data on the victim’s computer. Checked actions include the last access to the file, the time of modification of the registers, unclear entries in the logs, changes in the size of the file without it being open since the last use, and many more. The system can also investigate actions based on the so-called signature detection, that is, check sequences of behavior, the occurrence of which may indicate a potential attempted attack, for instance, several attempts to enter an incorrect password, or open a connection through a port of a previously blocked service. Currently, the HIDS solution is an addition to more complex protection packets. Examples of such programs include Tripwire [11], Aide [12], Afick [13] and IBM Proventia Server Intrusion Prevention System. There is also a combination of IDS and HIDS techniques, which means monitoring both the entire network and a single device. Such solutions are called NNIDS—network node IDS.

Manufacturers of commercial solutions for detecting attacks include Cisco IDS, IBM Proventia Server, Palo Alto. All these systems are based on databases of network threat signatures. This means that the threat in question must have already been checked by the research team of the security company. These solutions are not immune to new zero-day threats, that is, never-before-published information about bugs in the software. The exception is HP TippingPoint Next Generation Intrusion Prevention System (NGIPS). It is based on an in-depth analysis of packets and the behavior of devices within the network, but the company has not revealed the exact principle of the system for security and commercial reasons. However, we can observe that it does not follow only static rules like a typical firewall, and the decision to block traffic is made based on the identification of anomalies in the network.

The security of IT systems can be ensured in many ways. It is important that the systems are able to cooperate with each other without adversely affecting each other’s operations. User authorization based on their behavior is a technique that is commonly used in ICT systems. The development of this field is possible thanks to new data sets and the development of new methods of identification. An example is the

possibility of conducting authorization based on facial features—Face ID [14, 15] or spoken commands through voice analysis [16]. Despite a continuous improvement of the level of security measures used to store logins and passwords (e.g. Active Directory, Samba services) and the creation of security rules for tools such as firewalls based on specific IP addresses, it should be noted that infecting or hacking into a given station causes traffic to be generated from previously authenticated accounts and classified IP addresses. Therefore, these methods do not guarantee complete safety.

The first one is the analysis of data from network traffic. It has been the subject of many studies. The article [17] deals with the identification of network users based on the applications and thus the protocols they use. Data representing the problem was collected over a period of about two months. The set of obtained information occupied 112 GB and described the transmitted packets in text form (including port number, source and target address, number of transmitted packets and their size, time, etc.). Based on the data, a system of classical multilayer neural network type for recognizing 46 users was created. The efficiency of the presented solution ranges from 86.3% to 12.6%, with an average of 47.5%.

Also noteworthy is the work [18] using data from network traffic to detect whether users' computer hardware and phones were infected with malware. The study was conducted on data generated by 1923 users over a two-month period. They were characterized by 36 features, such as: the number of user records in the logs, the average length of session, the average time difference between the sessions starting times, the average number of sent and received bytes, the average number of sent packets, the number of unique IP addresses and ports in recorded traffic, the number of applications accessed by the user and the number of recorded sessions in specific periods. The purpose of the system was only to determine whether a given device was infected. The type of threat was not a subject of verification. The idea was to use the k-means algorithm [19] to group events. The final result of the correct threat classification was 79% of the test network traffic, which translated into 86% ROC AUC. area under the ROC curve) [20]. Another example of network traffic analysis using soft computing methods can be the application type detection based on transmitted packets. The authors of [21] proposed a method for packet analysis in a defined task this way, using convolutional networks and data collected in the ISCX VPN-nonVPN database [22]. The proposed solution can also be used for deep packet inspection (DPI) [23], a technique used to determine priority in network traffic or to block unauthorized applications in secured network traffic.

Information on the use of artificial neural networks in systems of network security can already be found in 2015 in the publication [24]. The system described there included a function for detecting network attacks. It was based on recurrent LSTM (long short-term memory) neural networks. The task was to identify threats in the shared KDD Cup 1999 [25] database, where four classes of attacks were listed: DoS, R2L, U2R, Probe, and 20% of data from normal traffic. This problem was solved with an accuracy of over 98%. A more effective solution to the above problems was presented in the fourth part of the monograph.

Designing neural networks to solve specific problems often comes down to modifying structures already well-known in the literature. Obviously, a model that is too small will not be able to capture all the important features during training. If a model is too large, a phenomenon of overfitting occurs [26, 89]. In order to improve the performance of the neural network in relation to both the quality of the model and the speed of operation, model simplification (pruning) is applied. It consists of reducing the parameters of the neural network by removing unnecessary neurons or weights. One of the methods for selecting an architecture by deleting nodes of a neural network, which is popular at the moment, is presented in [27]. In this method dropping neurons is based on the minimum value of the Euclidean norm (L2 norm) of weights vectors. A completely different approach is to reduce similar parameters of the model [28]. Neurons are compared during training and if the result of the action of two neurons for training data is the same, then just one neuron is left. This technique requires additional training after the parameter is removed. Optimization of neural network design and the training process itself is also part of the research undertaken in this book. In Chap. 5, a technique for optimizing neural networks with the help of so-called glial cells is presented.

The last chapter of this book describes the analysis of event logs of IT systems in order to detect hidden sequences using neural networks. In security-related tasks, detected anomalies in sequential data can be used, among other things, to detect intrusions [29]. The last chapter presents a technique based on the latest methods of processing text data, which allows for approximate processing using BERT networks. It can be applied to predict the possibility of failure of the inspected system.

Chapter 2

Artificial Neural Networks



Artificial neural networks are now one of the most popular tools for analyzing large data sets. They are the subject of extremely intensive research in the field of artificial intelligence. Advanced capabilities of these algorithms are used, for example, by Internet search engines. An example is the combination of Microsoft resources and Open AI to create a solution for the Edge browser that allows for intelligent data retrieval using ChatGPT.

The first authors who proposed the concept of building an artificial neuron in the form of an arithmetic-logical system were Warren McCulloch and Walter Pitts. The idea they presented in 1943 is still the basis for the construction of multilayer neural networks. Based on the formal model of the neuron, in 1957 Frank Rosenblatt presented a concept based on simple mathematical calculations performed by electromechanical elements—potentiometers, thanks to which the system was “able to learn”. It consisted of several neurons and constituted the first neural network. In the training process, the system adjusted the value of potentiometers to select the best values of weights in the network. Interest in neural networks waned when Marvin Minsky and Seymour Papert published “Perceptrons” in 1969 [30]. They pointed to the largest drawback of neural networks at that time, which stopped the development of this science for many years. Minsky and Papert proved that under the assumption of linear activation functions, it makes no sense to build structures consisting of more than one layer of neurons. Single-layer networks, however, are not able to model some mappings, e.g., the XOR (exclusive or) problem, see Fig. 2.1.

The change of activation functions to nonlinear and the introduction of a new technique for training neural networks—the algorithm for backpropagation of errors in 1975 by Paul Werbos [31] allowed for the construction of multilayer networks, now called Fully Connected layers (FC) or dense layers). At that moment, the foundations were laid for the construction of current architectures—the so-called deep networks. A diagram of a classical multi-layer network is shown in Fig. 2.2.

Its structure is based on the connection of many individual neurons N_i shown in Fig. 2.3. In this model, the symbol x_i denotes subsequent input signals, while w_i is the

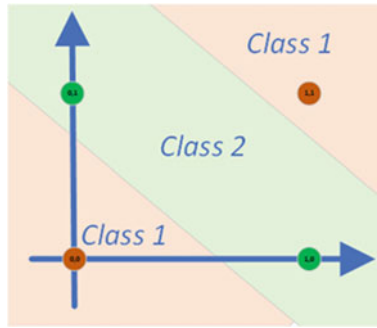


Fig. 2.1 The XOR problem—the solution of the separability problem in the case of samples positioned in this way is possible only using nonlinear activation functions

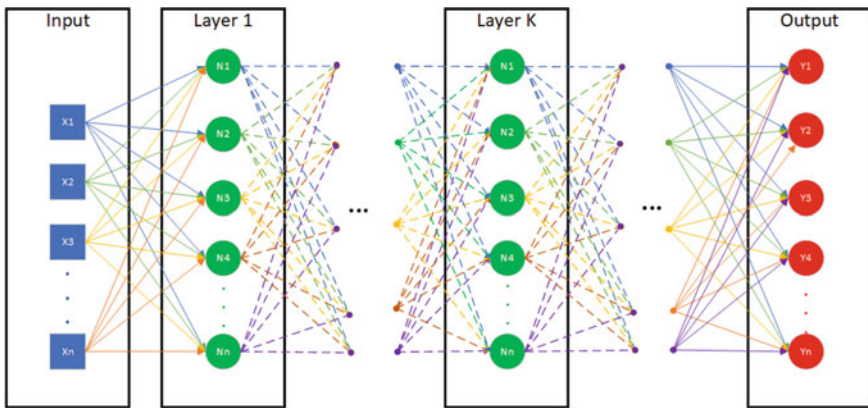


Fig. 2.2 Diagram of a multi-layer neural network

vector of weights in the i -th unit subject to adaptation in the training process. Dependencies that allow the training process to be performed according to the algorithm of backpropagation of errors were determined based on the book [1].

The output signal of the neuron number (i) in the layer number (k) is determined based on the formula

$$x_i^{(k)}(n) = +1$$

for $i = 0$ and $k = 1, L$. The input signal of the neuron N_i^k is related to the output signal of the $k - 1$ layer as follows

$$x_i^{(k)} = \begin{cases} u_i(n) & \text{for } k = 1 \\ y_i^{(k-1)}(n) & \text{for } k = 2, \dots, L \\ +1 & \text{for } k = 1, \dots, L \text{ and } i = 0 \end{cases} \quad (2.1)$$

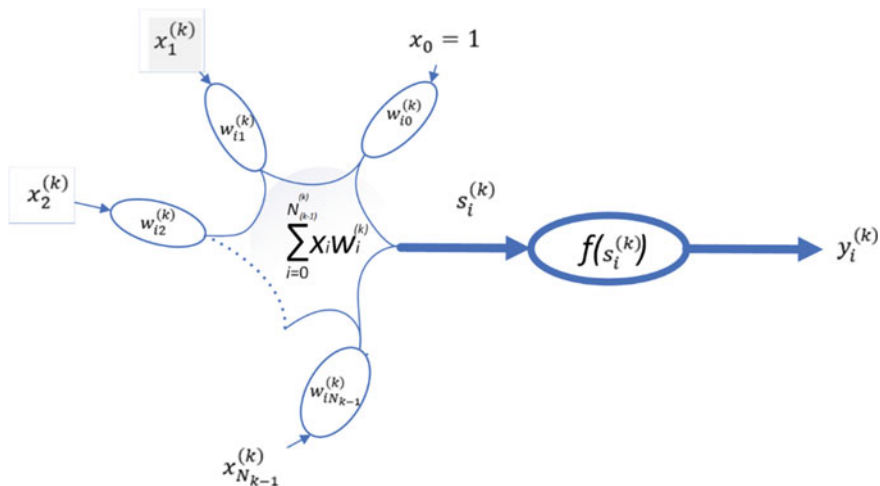


Fig. 2.3 Mathematical model of neuron

Figure 2.3 shows the action of a single neuron, where $w_{ij}^{(k)}(n)$ is the vector of weights, w_i is the weight of the i -th neuron $i = 1, \dots, N_k$, of the k -layer connecting the neuron to the j -th input signal $x_j^{(k)}(n)$, $j = 0, 1, \dots, N_{k-1}$. The vector of neuron weights N_i^k is denoted as follows

$$w_i^{(k)}(n) = \left[w_{i,0}^{(k)}(n), \dots, w_{i,N_{k-1}}^{(k)}(n) \right]^T, \quad k = 1, \dots, L, \quad i = 1, \dots, N_k \quad (2.2)$$

The output signal of the neuron N_i^k at the n -th moment, $n = 1, 2, \dots$, is described by the formula

$$y_i^{(k)}(n) = f\left(s_i^{(k)}(n)\right) \quad (2.3)$$

where

$$s_i^{(k)}(n) = \sum_{j=0}^{N_{k-1}} w_{ij}^{(k)}(n) x_j^{(k)}(n) \quad (2.4)$$

Based on the work [1], dependencies will be derived that allow for the adaptation of weights in neural networks in accordance with the algorithm of backpropagation of errors. Note that the output signals of neurons in the L -th layer

$$y_1^L(n), y_2^L(n), \dots, y_{N_L}^L(n) \quad (2.5)$$

are also the output signals of the entire network. They are compared with the so-called network template signals

$$d_1^L(n), d_2^L(n), \dots, d_{N_L}^L(n) \quad (2.6)$$

As a result, we get errors

$$\varepsilon_i^{(L)}(n) = d_i^{(L)}(n) - y_i^{(L)}(n), \quad i = 1, \dots, N_L \quad (2.7)$$

We can formulate the measure of an error resulting from the comparison of signals (2.5) and (2.6) as the sum of squares of the differences (2.8), i.e.

$$Q(n) = \sum_{i=1}^{N_L} \varepsilon_i^{(L)2}(n) = \sum_{i=1}^{N_L} \left(d_i^{(L)}(n) - y_i^{(L)}(n) \right)^2 \quad (2.8)$$

Formulas (2.3) and (2.4) indicate that the measure of error (2.8) is a function of network weights. Network training involves adaptive correction of all weights $w_{ij}^{(k)}(n)$ to minimize this measure. To correct any weight, we can apply the steepest descent rule in the form (2.9), where the constant $\eta > 0$ determines the size of the correction step.

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) - \eta \frac{\partial Q(n)}{\partial w_{ij}^{(k)}(n)} \quad (2.9)$$

If

$$\frac{\partial Q(n)}{\partial w_{ij}^{(k)}(n)} = \frac{\partial Q(n)}{\partial s_i^{(k)}(n)} \frac{\partial s_i^{(k)}(n)}{\partial w_{ij}^{(k)}(n)} = \frac{\partial Q(n)}{\partial s_i^{(k)}(n)} x_j^{(k)}(n) \quad (2.10)$$

and we assume that

$$\delta_i^{(k)}(n) = -\frac{1}{2} \frac{\partial Q(n)}{\partial s_i^{(k)}(n)} \quad (2.11)$$

we will obtain:

$$\frac{\partial Q(n)}{\partial w_{ij}^{(k)}(n)} = -2\delta_i^{(k)}(n)x_j^{(k)}(n) \quad (2.12)$$

The final form of the algorithm is:

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) + 2\eta\delta_i^{(k)}(n)x_j^{(k)}(n) \quad (2.13)$$

The method of calculating the value of $\delta_i^{(L)}(n)$ given by formula (2.16) depends on the layer. For the last layer, it is

$$\begin{aligned}\delta_i^{(L)}(n) &= -\frac{1}{2} \frac{\partial Q(n)}{\partial s_i^{(L)}(n)} = -\frac{1}{2} \frac{\partial \sum_{m=1}^{N_L} \varepsilon_m^{(L)2}(n)}{\partial s_i^{(L)}(n)} = -\frac{1}{2} \frac{\partial \varepsilon_i^{(L)2}(n)}{\partial s_i^{(L)}(n)} \\ &= -\frac{1}{2} \frac{\partial \left(d_i^{(L)}(n) - y_i^{(L)}(n) \right)^2}{\partial s_i^{(L)}(n)} = \varepsilon_i^{(L)}(n) \frac{\partial y_i^{(L)}(n)}{\partial s_i^{(L)}(n)} \\ &= \varepsilon_i^{(L)}(n) f' \left(s_i^{(L)}(n) \right)\end{aligned}\quad (2.14)$$

For any layer $k \neq L$ we obtain

$$\begin{aligned}\delta_i^{(k)}(n) &= -\frac{1}{2} \frac{\partial Q(n)}{\partial s_i^{(k)}(n)} = -\frac{1}{2} \sum_{m=1}^{N_{k+1}} \frac{\partial Q(n)}{\partial s_m^{(k+1)}(n)} \frac{\partial s_m^{(k+1)}(n)}{\partial s_i^{(k)}(n)} \\ &= \sum_{m=1}^{N_{k+1}} \delta_m^{(k+1)}(n) w_{mi}^{(k+1)}(n) f' \left(s_i^{(k)}(n) \right) \\ &= f' \left(s_i^{(k)}(n) \right) \sum_{m=1}^{N_{k+1}} \delta_m^{(k+1)}(n) w_{mi}^{(k+1)}(n)\end{aligned}\quad (2.15)$$

Let's define the error in the k -th layer (except for the last layer) for the i -th neuron

$$\varepsilon_i^{(k)}(n) = \sum_{m=1}^{N_{k+1}} \delta_m^{(k+1)}(n) w_{mi}^{(k+1)}(n), \quad k = 1, \dots, L-1 \quad (2.16)$$

When we substitute (2.21) into Formula (2.20), we obtain

$$\delta_i^{(k)}(n) = \varepsilon_i^{(k)}(n) f' \left(s_i^{(k)}(n) \right) \quad (2.17)$$

As a result, the algorithm of backpropagation of errors can be formulated as follows

$y_i^{(k)}(n) = f \left(s_i^{(k)}(n) \right), \quad s_i^{(k)}(n) = \sum_{j=0}^{N_{k-1}} w_{ij}^{(k)}(n) x_j^{(k)}(n)$	(2.18)
$\varepsilon_i^{(k)}(n) = \begin{cases} d_i^{(L)}(n) - y_i^{(L)}(n) & \text{dla } k = L \\ \sum_{m=1}^{N_{k+1}} \delta_m^{(k+1)}(n) w_{mi}^{(k+1)}(n) & \text{dla } k = 1, \dots, L-1 \end{cases}$	(2.19)
$\delta_i^{(k)}(n) = \varepsilon_i^{(k)}(n) f' \left(s_i^{(k)}(n) \right)$	(2.20)
$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) + 2\eta \delta_i^{(k)}(n) x_j^{(k)}(n)$	(2.21)

The above table defines the dependencies that are necessary to train the network using the backpropagation of errors algorithm. The error signal ε is propagated from the last to the first layer and based on it the value of the weights in the network is modified.

The above algorithm can be extended with additional techniques that increase the training speed and improve the training quality of the model. A widely used method is *momentum*, described by Formula (2.22).

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) + 2\eta\varepsilon_i^{(k)}(n)f'(s_i^{(k)}(n))x_j^{(k)}(n) + \alpha[w_{ij}^{(k)}(n) - w_{ij}^{(k)}(n-1)] \quad (2.22)$$

This technique involves introducing an additional element to the recursion, where the parameter $(0,1)$.

2.1 Initialization of Weights Values in Neural Networks

Initialization of neural network weights before the start of the training process is of great importance for training efficiency and effectiveness. Correct initialization of weights allows the training process to converge more quickly and can help avoid problems such as stopping at local minima or slow learning. The most popular method is the random initialization of weights according to a certain distribution, for example, a uniform distribution or a normal distribution. In the early research on neural networks, it was assumed that weights should be assigned pseudorandom values within the range $(-1,1)$. One of the first methods for initializing neural network parameters is the method of scaling the values of weights relative to the size of the network input [2] (2.25)

$$w_{ij} \sim U\left(0, \frac{1}{N_k}\right) \quad (2.23)$$

where U denotes a uniform probability distribution of values from the range $(-1,1)$, N_k is the size of the input layer and k is the number of the network. An extension of the above is a method presented in [3]. This method is based on the average number of connections located in two adjacent layers of a neural network.

$$w_{ij} \sim U\left(-\sqrt{\frac{6}{N_k + N_{k+1}}}, \sqrt{\frac{6}{N_k + N_{k+1}}}\right) \quad (2.24)$$

This technique involves introducing an additional element to the recursion, where the parameter $\alpha \in (0, 1)$.

2.2 Convolution Neural Networks

The recent huge interest in neural networks—which can be called a revolution in research on AI techniques—has been caused by the development of a new type, the so-called convolutional neural networks [4]. Unfortunately, the models described earlier had a number of limitations [5]. Convolutional networks have a much more advanced structure and much greater potential in terms of operation. The basis of their functioning is convolutions. A diagram of these operations is presented in Fig. 2.4.

Multi-layer networks described in the previous section Feedforward neural networks (FNN) are commonly used for processing data with fixed input sizes, i.e. where each observation consists of a set of features with a size defined at the design stage. They are used for a variety of tasks, including: classification, regression or even generating new data (patterns) [6–8]. They can be successfully used in areas such as recognizing handwritten numbers, analyzing sentiment in text, or identifying objects based on a set of features. However, they have significant limitations when working on large-scale information or on a structure in which key objects can be rotated or scaled. Examples of such data sets are images. In such cases, convolutional neural networks (CNN) are a much better solution. The architecture of these networks allows us to use feature maps to extract local patterns and features. In addition, empirical studies clearly confirm the advantages of convolutional networks when processing scaled and rotated objects. A breakthrough in the works related to CNN networks is described in numerous scientific publications [9–12].

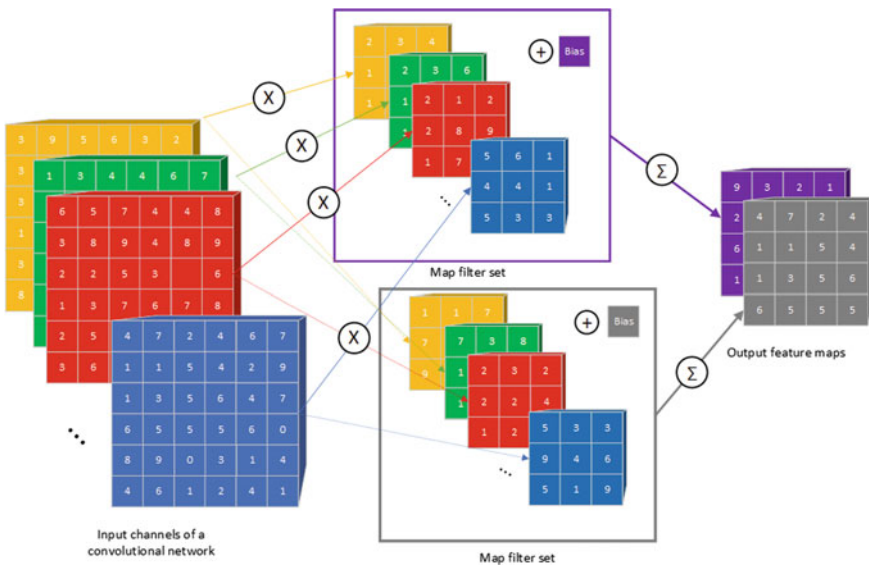


Fig. 2.4 Examples the max-pooling and sub-sampling operations with a size of 2×2

Convolution is the basic operation in data analysis performed by convolutional networks. During this operation, a filter with specific weights of neurons is “shifted” over the input data, and then the scalar product between the filter and the data fragment is calculated. This process leads to the extraction of local patterns and features from the data. The values of individual filter elements determine the weights of the neurons. The result of a convolution operation is a feature map, representing extracted features and patterns. Figure 2.4 presents how this operation works.

The convolution operation described above can be written using Formula (2.25), where \mathbf{I} stands for the input image (pixel matrix), \mathbf{F} for filter with a dimension of $f_1 \times f_2$ (neuron weight values), and b for bias, \tilde{n} for the number of maps of input features on which the convolution is performed.

$$(\mathbf{I} * \mathbf{F})_{ij} = \sum_{m=0}^{f_1} \sum_{n=0}^{f_2} \sum_{c=1}^{\tilde{n}} \mathbf{I}_{i+m,j+n,c} \mathbf{F}_{m,n,c} + b \quad (2.25)$$

The key element in the convolutional layer are filters values. They are selected in the training process of the neural network using the previously presented error backpropagation algorithm.

In the literature, we can distinguish three types of filters applied in neural networks, depending on their dimensions: 1D, 2D, and 3D. Each of them is used in different areas of information processing. 2D and 3D filters are particularly useful in neural networks designed for image processing, while 1D filters are used in natural language processing (NLP) and signal processing, where data sequence is important.

3D filters are extensions of 2D filters that take into account the third dimension, as in the case of image sequence analysis (e.g., video time sequences) or analysis of three-dimensional objects. 3D filters are used in medicine, computer graphics, gesture recognition, spatial data processing, and other fields where spatial information is crucial.

1D filters are used in data processing where data is ordered or sequential, e.g. in the analysis of texts, audio signals or temporal data. Examples of sequential processing applications include speech recognition, text data analysis, acoustic signal processing, and many more [13]. Depending on the type of data and the context, the right filter selection (1D, 2D, 3D) allows us to effectively model the features and patterns present in the data, taking into account their specifics and structure.

In addition to the operation of convolution, working with convolutional networks may also entail pooling [14], padding [15] and sub-sampling [16] (average pooling [17]) operations, example in Fig. 2.5. The pooling layer is used mainly after the convolutional layer and is intended to reduce the spatial dimensions of data. The most common technique is the so-called max pooling, although other types exist, including average pooling or L2 pooling. The max pooling operation consists of dividing the input data into areas (e.g., square regions) and selecting the largest value in each of them. Thus, this operation extracts the most important information from a given area and reduces its dimensions. Pooling helps maintain local invariance,

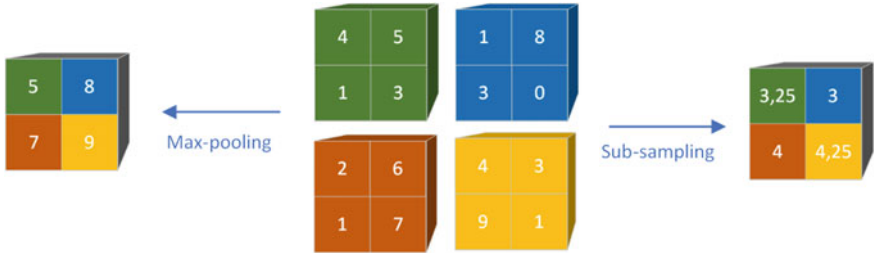


Fig. 2.5 Examples the max-pooling and sub-sampling operations with a size of 2×2

which means that regardless of the location in the input data, the isolated feature will be recognized.

Pooling is performed in the process of modeling the structure of the neural network by adding appropriate layers. It helps to reduce the dimensionality of data, and thus the number of parameters and calculations in the network. This, in turn, speeds up training and reduces the risk of overfitting. Another operation characteristic of convolutional networks is sub-sampling. It enables the recognition of features at different scales and reduces the impact of small shifts in data. It should be emphasized here the sub-sampling and max-pooling operations not only reduce the dimensionality of data but also help to extract the most important features, which can lead to better performance and increase in the ability of networks to generalize knowledge. These techniques are often used in convolutional networks, both in convolutional and fully connected layers, to reduce the size of data while preserving relevant information and improving network performance.

It should be noted that each of the above operations in convolutional networks affects the change in the dimensions of the signal in subsequent layers (dimensions of the so-called feature maps—FM). Apart from determining the dimensions of the input signals in the training process, it is also necessary to determine the step \hat{s} of “shifting” the filters. This value is calculated using Formulas (2.26), (2.27)

$$\hat{w}' = \frac{\hat{w} - F_{\hat{w}}}{\hat{s}} + 1 \tag{2.26}$$

$$\hat{h}' = \frac{\hat{h} - F_{\hat{h}}}{\hat{s}} + 1 \tag{2.27}$$

where \hat{h}' , \hat{w}' denotes new dimensions of feature maps after max-pooling or sub-sampling performed using a filter with the size of $F_{\hat{w}} \times F_{\hat{h}}$. When designing a network, we need to select the dimensions of the filters in subsequent layers so that the resulting feature maps will allow us to perform subsequent convolution or pooling. Obviously, the values determining the height and width must be integers, hence the results obtained from the calculations using Formulas (2.26), (2.27) must be rounded up.

Based on the assumptions resulting from Eqs. (2.26) and (2.27), the layer size is reduced in each operation. This is an additional difficulty when designing convolutional networks. A popular solution to this problem is to perform a padding operation on the input layer. The padding layer (operation) is used to preserve information at the edges of input data during the convolution. As a result, the size of the output may decrease depending on the size of the filters and the shift step. Padding consists in adding values equal to “0” around all edges of the input data. For example, if we use padding of size one, we “expand” the range of input data by one row of zeros at the top and bottom, and one zero column on the left and right side of the data. Thanks to this, convolutions can be performed on the input data while the original information on the edges is preserved. It should be emphasized that padding is especially relevant when important features of objects are located at the edges of the input data, or when we want to maintain the size of the input and output data in the machine-learning process (Fig. 2.6).

The biggest advantage of convolutional networks is the lower number of weights and connections compared to fully-connected networks. This is achieved through the mechanism of shared weights. It is a technique used in convolutional networks, where the same weights are used to process different fragments of input information.

In practice, filters (weights) in convolutional layers are applied over the entire input data space. A good example is the processing of image data, where filters are “shifted” (convolution operation) along the entire content of the image. The convolution network is then able to detect the same features and patterns in different spots of the image. There are several advantages to weight sharing. As mentioned earlier, it significantly reduces the number of parameters in the network, which helps to limit the computing load on the equipment and prevents overfitting. Another benefit is that parameter sharing allows for better use of spatial information, as the same weights are applied to different parts of data. Weight sharing is an important mechanism in convolution networks that improves the efficiency of parameters use and the extraction of features from data. Noise and interference in images pose a significant challenge to convolutional networks. As mentioned earlier, one of the main types of information processed by deep networks are images. Although the human eye may

Fig. 2.6 Example of padding process (dark green indicates original data, light green—added zero value vectors)

0	0	0	0	0
0	1	1	5	0
0	1	3	5	0
0	6	5	5	0
0	0	0	0	0

be able to ignore subtle changes in images, in neural networks even minor disruptions can affect the classification process and lead to errors. Convolutional networks, built of convolutional layers, have the ability to detect local patterns and features in images. Convolutional filters act as detectors of patterns, such as edges, textures or shapes. Detected features are then used to classify objects. However, noise in images can result in unwanted interference in the detected features, making the classification process difficult. For example, small changes in brightness, contrast or extra pixels can affect the final results of classification by a convolutional network [18]. Although the change in the image may be invisible to humans, the convolutional network may not be able to correctly identify the objects. This may be because the noise introduces unexpected changes in the features detected by the convolution filters, making it difficult for the network to interpret them correctly. To deal with this problem, many techniques for improving the noise immunity of convolutional networks can be applied. Examples include normalization layers that help reduce the impact of noise, and data augmentation techniques that introduce a variety of transformations to training data to increase the generalization capacity of the network. Another approach is to use net architectures with a greater ability to model noise and interference. For example, residual layer networks (ResNet) or generative networks (GAN) [19] may be more resistant to interference, which results in better classification outcomes in the presence of noise. In any case, ensuring the noise immunity of convolutional networks is an important research challenge that leads to the continuous development of techniques and methods to improve the performance of these networks under realistic conditions associated with the occurrence of image interferences.

Chapter 3

Application of Artificial Intelligence Methods in Profiling Computer Network Users



The use of artificial intelligence (AI) techniques in the field of IT security is becoming increasingly common. Taking into account the capabilities of convolutional networks in the field of analysis of large data sets described in previous sections, it is a natural approach. Soft computing algorithms offer new possibilities to detect, analyze and prevent various types of threats and cyber attacks. Here are some important aspects related to the use of AI in IT security:

1. Threat detection and analysis: AI can be used to analyze large amounts of data concerning network activity to detect unusual patterns and behaviors that may indicate potential threats. Advanced machine training algorithms can identify previously unknown attacks and adapting to new types of threats.
2. Prevention of attacks: AI can help build defence systems that are able to recognize and block attempted attack in real-time. The use of machine learning to analyze and classify new types of attacks allows for rapid response and implementation of appropriate preventive measures.
3. Log analysis and incident detection: AI can automate the analysis of large sets of system, network, and application logs to identify irregularities, suspicious activity, and potential security incidents. This can significantly shorten the response time and allows for appropriate corrective action to be taken quickly.
4. Protection against phishing and spam: AI can be used to identify and block phishing attacks and spam. Advanced machine learning models can learn to recognize the features and patterns that indicate these types of attacks, which enables effective filtration and protection against them.
5. Analysis of user behavior: Using AI to analyze user behavior can help detect unauthorized access and suspicious activity. Systems can monitor user activity, identify non-conformities with normal behavior patterns and suspicious activity, which enables a rapid response to potential threats.

The introduction of artificial intelligence into IT security has the potential to significantly increase effectiveness in detecting and preventing attacks, and speed

up the response to incidents. However, independently of these advantages, it is also necessary to provide appropriate protective measures, monitor the machine learning process and understand the limitations and potential risks associated with the use of AI in this area. This section presents proprietary solutions for creating user profiles in computer networks. One of the key pieces information that are processed by neural networks in this approach are logs of user activity in the computer network, including the history of visited websites. It is worth noting that building a correct user profile can allow us to find data that deviate from the constructed patterns and thus detect anomalies. Non-standard behavior in the computer network, which differs from that defined in the profile, can be caused by the following factors:

1. hacking into the IT system and generating traffic by an unauthorised person;
2. presence of viruses or trojans that generate network traffic on infected stations.

Analyzing user logs and comparing them with their profile can help us identify such anomalies and take appropriate action. Applying the newest neural network models described in the literature to analyze user logs and create their profiles enables us to effectively detect suspicious activity and anomalies in the computer network [20]. This in turn makes it possible to react quickly to potential threats and introduce appropriate protective measures.

An example of a user profile construction application is [21], based on smartphone user activity. Research conducted on collected data showed that each user has a unique way of using a mobile device. This is due to their individual preferences, habits and physical features. The concept of this system is analogous to the method used in criminology by profilers to identify the perpetrators [22]. Profiles try to discover the modus operandi by analyzing collected the information and determining features that are characteristic of only one person. Each user of the network leaves traces of their activities. Some of them are consciously generated by the user, for example, signed posts on social media, while others are closely related to the mechanisms of operation of computer networks. However, it should be remembered that any such activity on the network can be reflected in various logs. Currently, administrators have access to a huge amount of information that can be analyzed. Obviously, analyzing single logs will be less effective than an analysis of long-term events. It should be stressed that this task is practically impossible to be performed by a human without appropriate tools. An important part of current research is aimed at determining what features should be taken into account in order to construct the best user profile. The concept of a user profile refers to a set of features and their values, based on which it is possible to identify a particular person who uses the computer network. Examples of features for network traffic include: the type of applications used (TCP ports, target addresses, domain names), the moment at which requests are executed, the domain names of visited websites and their content.

Based on the studies described in this book, it can be concluded that an appropriate combination of characteristic values is a kind of “fingerprint” of the user. However, it is important to understand that the resources of the Internet are constantly evolving in a dynamic way. Therefore, the system based on the described solution must constantly collect new data, and already created profiles must be regularly

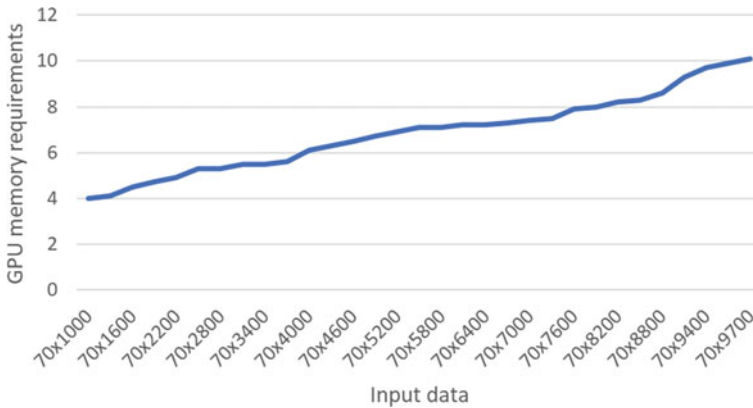


Fig. 3.1 Memory usage chart for a convolutional neural network with varying input data using a minibatch of size 260

updated. The proprietary uses real logs from a cluster of Palo Alto tools, working in the active-active mode, which contain information about network traffic that uses the HTTP and HTTPS protocols. One of the key pieces of information processed in this system will be URLs. Requests are generated when network users access the Internet via browsers, but also by various applications running on a given station (for example, antivirus programs or system updates). Collected features, such as URLs of websites visited by the users, are subjected to pre-processing that makes them usable for machine learning algorithms.

In the presented system, URLs are grouped into sessions of 8 to 300 addresses. The limitation of the maximum number of addresses per session is due to hardware limitations associated with training convolutional networks. The input data for the neural network is encoded using numerical vectors. Therefore, it is necessary to translate text information (e.g. URLs) into numeric values. This translation process is done by “1 of n ” encoding [23]. Assuming an average URL length of 32 characters and a dictionary size of 70 characters, the neural network input layer must be 70×9600 . This represents the maximum memory requirement of GPU tools.

Figure 3.1 shows the actual memory requirements of GPU cards depending on the size of the URL sequence visited. The simulation process uses the Microsoft CNTK library and the CUDA version 10 controller.

Apart from the maximum number of URLs per session, another criterion for grouping URLs is time. It was assumed that if more than 30 min have passed between two registered events for a given user, the URLs belong to separate sessions. This suggests that the user did not use Internet resources in the form of websites (most likely, they left the computer).

In Fig. 3.2, you can see the trace of events recorded at a given moment on the PaloAlto cluster. Web traffic in the form of m in visited websites in real conditions was used for the research described in this book. All data collected in the Palo Alto device logs were divided into test and training sets with a ratio of 20:80.

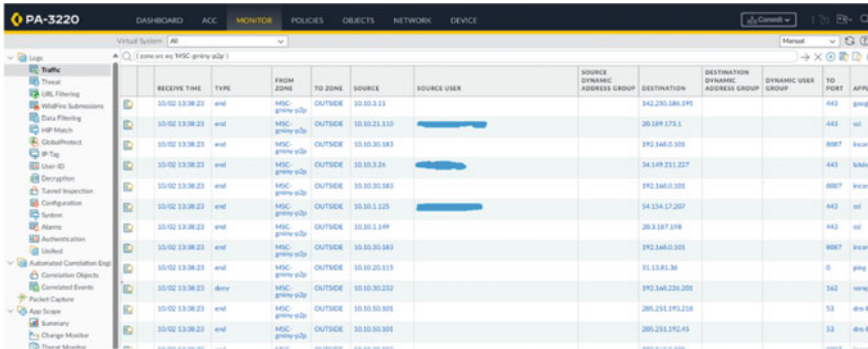


Fig. 3.2 Example of network traffic logged on the Pa3220 family PaloAlto cluster

Table 3.1 TOP most frequent IP addresses present in the training data

IP	Count	Domain
142.250.217.100	24,810	Google LLC
20.76.201.171	22,158	Microsoft Corporation
142.250.203.142	16,922	youtube.com
212.77.98.9	14,982	Wirtualna Polska Media S.A
172.217.22.3	14,112	Google LLC
86.111.241.163	10,118	elara.iq.pl
185.184.8.30	9186	PHOENIX NAP
173.241.240.143	8125	OPENX TECHNOLOGIES
217.74.66.216	6267	INTERIA.PL Sp z.o.o
185.14.253.220	6731	s11.smartsupp.com

Tables 3.1 and 3.2 present the most popular addresses in the test and training sequences used for conducting research. After conducting the training and testing process on this prepared data set, the received solution was to be tested in real-life conditions. About three months after the completion of the system, new sets of information from the Palo Alto firewall were collected and new data was created to verify the proposed neural network model. This prepared validation set served to assess the quality of the system operation.

Table 3.2 TOP most frequent IP addresses present in the test data

IP	Count	Domain
142.250.217.100	10,754	Google LLC
142.250.203.142	8214	youtube.com
16.58.208.46	8110	Google LLC
20.76.201.171	6234	Microsoft Corporation
212.77.98.9	5104	Wirtualna Polska Media S.A
172.217.23.174	4174	Google LLC
157.240.253.35	4017	facebook.com
212.77.98.29	2918	o2.pl
217.74.66.216	2853	INTERIA.PL Sp z.o.o
185.14.253.220	1428	s11.smartsupp.com

3.1 Profiling Users Using the Random Forest Algorithm

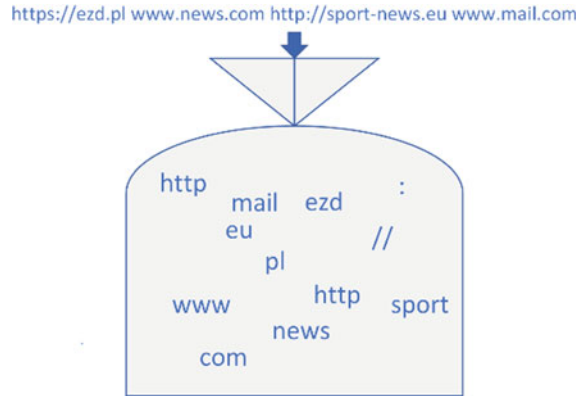
The random forest algorithm is one of the most popular machine learning algorithms in the family of ensemble methods. It is used for both classification and regression tasks.

The idea behind a random tree algorithm is to construct multiple decision trees and combine them into a single model. Each tree is trained on randomly selected subsets of training data, as well as on randomly selected subsets of features. Such random sampling is intended to increase tree diversity and reduce overfitting. The process of building a tree in a random tree algorithm consists in dividing data based on different features, so as to maximise the purity of the division (e.g. entropy, Gini index). Trees are built until a certain stop criterion is reached, for example, the maximum depth of the tree or the minimum number of observations in the leaf. Once trees are constructed, predictions are made by majority voting (in the case of classification) or averaging (in the case of regression) the results of all the trees in the forest. The random tree algorithm has many advantages, such as resistance to overfitting, the ability to handle a large number of features and large data sets, and the ability to evaluate the validity of features. However, the random tree algorithm may be prone to overtraining in case of improper selection of hyperparameters. In these situations, applying regularisation techniques, such as reducing tree depth or increasing the number of trees in the forest, can help improve results.

To apply this algorithm, it is necessary to write the input data in an appropriate way. In this solution, it will be combined with the Bag-of-Words (BOW) method [24]. BOW is a natural language processing technique used in text analysis and classification tasks [25]. It consists in representing a document as a collection of unique words, while ignoring their order, grammatical structure and context.

The process of creating a BOW representation consists of several steps. Figure 3.3 illustrates an example of creating a BOW (Bag of Words) set from URL addresses. First, tokenization, i.e. splitting the document into single words or tokens, is

Fig. 3.3 Example of using the BOW method to build learning sets



performed. Then a dictionary containing all unique words that occur in all documents is created. The next step is to assign numeric values to each word in the dictionary, e.g., by assigning indexes. Ultimately, each document is represented as a vector of length equal to the number of words in the dictionary. Values in a vector can represent, for example, the number of occurrences of a word in a document or its validity in the context of an entire document collection (for example, TF-IDF-term frequency-inverse document frequency). TF-IDF [26] is a statistical method used in natural language processing to evaluate the validity of words in a document in the context of an entire document collection. It is widely used for text analysis, information retrieval and document classification. TF-IDF takes into account the two main features of a word: term frequency (TF) in a document and inverse document frequency (IDF) throughout a document collection.

Term frequency (TF) measures how often a word appears in a document. It is calculated as the ratio of the number of occurrences of a word to the total number of words in the document. It can be calculated in various ways, for example, using a normal frequency, a normalised frequency, or a logarithmic frequency scale.

IDF measures how rare a word is in a collection of documents. Words that occur less often have a higher IDF value. IDF is calculated as inverted frequency of occurrence of a word in the entire document collection. It can be calculated in a variety of ways, for example, using a simple inverse ratio of the frequency of a word to the number of documents containing that word, or more advanced measures such as IDF with smoothing. TF-IDF is calculated by multiplying the frequency (TF) of a word in a document by the “rarity” (IDF) of that word throughout the document collection. The resulting TF-IDF values indicate how important a given word is in a given document in the context of other documents. The use of TF-IDF allows us to discover keywords that distinguish a given document from other documents. It can be used to evaluate document similarity, categorize documents, search for information, or recommend content. It is worth noting that TF-IDF is one of many methods of assessing the validity of words within text analysis. There are also other techniques,

such as word embedding or methods based on machine learning, that can provide more complex word representations and better results in some text analysis tasks.

Although the BOW representation is a simple approach, it does have some limitations. It ignores the structure and semantics of the text and does not take into account the relationship between words—just the presence or absence of a given word in a given collection. In addition, for large data sets, it can lead to the creation of long vectors and sparse matrices, which in turn, may require a lot of memory and affect computational performance when processing such representations. To correct these shortcomings, more advanced word processing techniques are used, such as models based on neural networks (e.g. recurrent networks, attention models), or methods using word embedding.

The input features used in the proposed algorithm are the components of the URL of a particular user. In this case, special characters such as [.,: ,, ” /] were treated as a separator between words in accordance with the logic of URL construction and the BOW algorithm. In order to improve the performance of the algorithm, we should try to determine the impact of each element from the training and testing string using TF-IDF. In this approach, the weight of each word in the dictionary is determined based on the number of its occurrences in the entire training data set. Thanks to the use of TF-IDF, it becomes possible to determine the minimum weight value for each of the words used for training the classifier.

$$TF_{i,j} \cdot IDF_i = \frac{n_{i,j}}{\sum_k n_{k,j}} \cdot \ln\left(\frac{d}{m_i}\right) \quad (3.1)$$

Thus, this method allows us to significantly reduce the size of the dictionary used in the process of building training vectors (we remove elements with small weight values). Thus, the calculation time can be significantly reduced. The significance value of each element is calculated according to Formula (2.27), where $n_{i,j}$ is the number of occurrences of the pseudo-word i in the sequence j , where k is the number of all sequences in the training string, m_i is the number of sequences containing at least one pseudo-word i . d is the number of all sequences. The term pseudo-word refers to strings that build a URL, separated by characters from the previously defined list of separators. It was a reference to the basic terminology of the BOW method. In an experiment using these techniques to classify users, algorithms implemented in scikit-learn [27] and Python were applied. As part of the experiment, a dictionary of 531,841 pseudo-words, corresponding to parts of URLs, was created. The next step was to create a set of teaching vectors to represent the websites visited by the user in subsequent sessions. Based on that 16–512 decision trees were generated. Table 3.3 presents the results of studies for selected numbers of trees in a random decision forest. The values shown were averaged based on a tenfold repetition of each example. The following measures were used to compare the obtained results: Accuracy, Precision, Recall, and F1. Measures were calculated on the basis of [28, 29] using the scikit-learn package. The metric calculation for each class was performed using the mean.

Table 3.3 Correctness of users classification using the random forest algorithm

Number of trees in the forest	Accuracy (%)	Precision	Recall	F1	Training time [s]
16	31.8	0.21	0.21	0.21	8
32	38.1	0.28	0.25	0.28	10
64	48.3	0.51	0.47	0.48	15
128	68.1	0.71	0.61	0.61	29
256	69.1	0.71	0.63	0.62	72
512	67.2	0.69	0.60	0.60	98

The result of the designed system in the case of classification of 68 users was 69.1%. This result was obtained using 256 decision trees.

Figure 3.4 visualizes the data collected in Table 3.3. It shows how the learning time increases with the growth in the number of trees. Conducted experiments have shown that a larger number of trees does not significantly influence the increase of correctness of the classification, but it does significantly increase the training time of the model. The optimal result was achieved using this number of trees, which provided a balanced compromise between classification accuracy and training time.

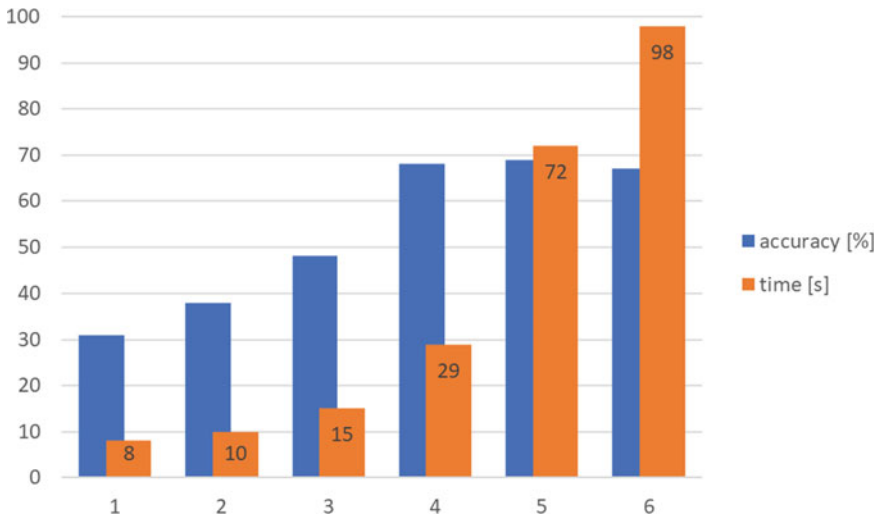


Fig. 3.4 Comparison of time and efficiency of proposed decision tree models

3.2 Convolutional Networks in User Profiling

Another approach to creating user profiles is based on convolutional networks, which were originally designed to classify images. To perform the task of building profiles, it is necessary to correctly encode input data that are not graphic objects. In [65], a method allowing for the processing of text data by convolutional structures was proposed. In the conducted study—as in the case of decision trees—the 1 of n encoding was used, where n represents the size of the dictionary and equals 70 characters. This dictionary contains alphanumeric characters (a – z), special and punctuation characters, and characters often found in URLs. According to this method and its assumptions, each user session is represented as a 70×9600 matrix, the values belong to the range $\{0,1\}$ and one-hot encoding at the character level is applied.

The one-hot encoding method, also known as binary category encoding, is a technique used in data analysis and machine learning to represent categorical or discrete variables as binary vectors. The idea behind this method is to assign each unique category a separate dimension or feature. Then, for each observation, a vector of length equal to the number of unique categories is created, in which all values are zero, except for one dimension corresponding to the observation category, which has a value of 1. For example, if we have a categorical variable “color” with three unique values: “red”, “green” and “blue” we can encode it using three dimensions: [1, 0, 0] for “red”, [0, 1, 0] for “green” and [0, 0, 1] for “blue”. One-hot encoding is widely used because it allows categorical variables to be represented in a format that can be directly processed within machine learning models. It also allows us to preserve information about the distance between categories, without introducing an artificial order. The disadvantage of such information encoding is that it leads to an obvious increase in the dimensionality of data, especially when a large number of unique categories/classes are being considered. To counteract this, we can introduce additional operations aimed at reducing dimensionality, such as principal component analysis (PCA) or feature selection. The following is an example of “1 of n ” encoding mechanism for two sentences: “ This is an example of the first sentence” and “This is an example of the second sentence” First, we create a dictionary of unique words: [“This”, “is”, “an”, “example”, “of”, “the”, “first”, “second”, “sentence”]. Then, we encode sentences using the one-hot method:

The first sentence now has the form: [1, 1, 1, 1, 1, 0, 1] and the second sentence has the form: [1, 1, 1, 0, 1, 1].

It is worth noting that one-hot encoding for texts often leads to the formation of sparse matrices, especially with very large dictionaries. As a result, numerous performance problems may arise, which, as mentioned earlier, require the use of techniques to reduce dimensionality. One-hot encoding on the level of characters consists of representing each character in the text as a separate one-hot vector. In this case, each character is assigned a unique index in the alphabet and is then represented as a vector of length equal to the size of the alphabet. For example, consider the word “coding”. We create an alphabet consisting of unique characters: [‘C’, ‘o’, ‘d’, ‘i’, ‘n’, ‘g’]. Then we encode the word using the one-hot method at the character level:

'C' → [1, 0, 0, 0, 0, 0] 'o' → [0, 1, 0, 0, 0, 0] 'd' → [0, 0, 1, 0, 0, 0]
'i' → [0, 0, 0, 1, 0, 0] 'n' → [0, 0, 0, 0, 1, 0] 'g' → [0, 0, 0, 0, 0, 1]

As a result, each character is represented as a vector with a length equal to the number of unique characters in the alphabet, where only one position has a value of 1, and the rest of the positions have a value of 0. In the context of the defined problem, where the input data are encoded sessions of visited URLs, a convolutional network structure was designed. Its main aim was to classify these sessions and assign them to specific users.

For the purpose of the task, a convolutional network structure consisting of several layers was used. It included convolutional layers, nonlinear activation layers (e.g. ReLU), and pooling layers that reduce the size of data. The adopted network architecture is represented in Fig. 3.5. In the classification process, the input data, which are encoded sessions of visited URLs, are processed by successive layers of the convolutional network. Convolutional layers extract local features from the input data while pooling layers reduce the dimensions of the data by selecting the most important features. The last layer in the tested model is a fully connected layer, which ultimately presents the classification result for a given session—and assigns a user ID.

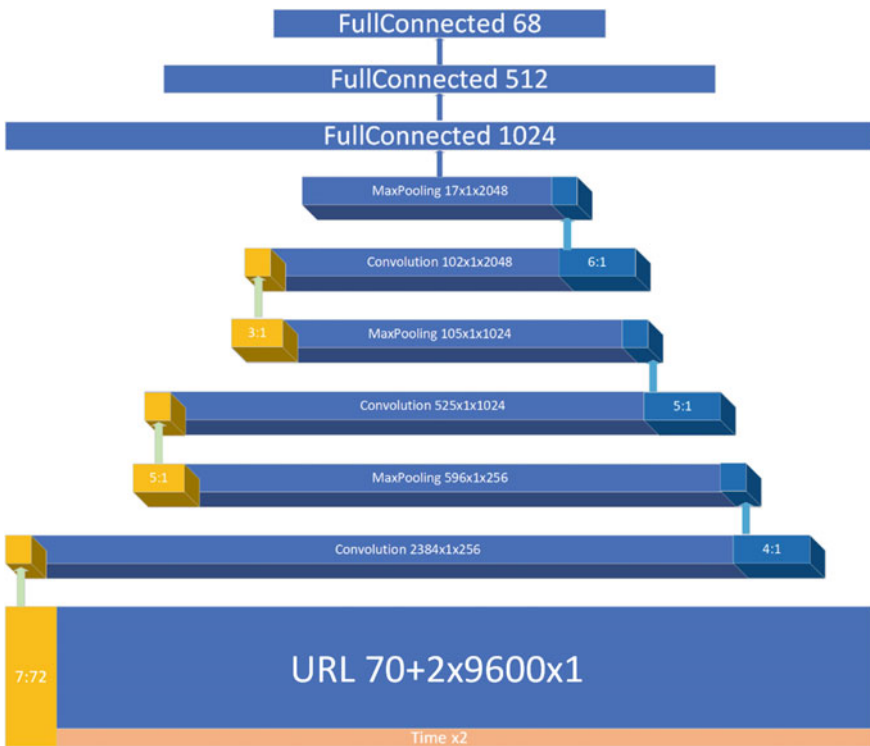


Fig. 3.5 Diagram of a network used for user profiling

The network input, in addition to data in the form of encoded visited URLs, was further expanded by two channels that represent the time of day when the user accessed the network resources. This information provides a time context for the classification process. In this case, each session consisting of visited URLs is now represented by a 9600×72 matrix, where 70 columns correspond to Bag-of-Words encoding for URLs, and two additional columns represent the time of day. A diagram of the network structure including these additional channels is shown in Fig. 3.5. Additional channels representing the time of day were encoded so that the value 1 for channel 71 corresponds to the hours when the user is most likely at work, that is between 7:30 a.m. and 4:30 p.m., and the value 1 for channel 72 corresponds to the remaining hours. Thanks to the introduction of this information, the variability of user behavior depending on the time of day could be analyzed, which has a significant impact on the quality of network operation (quality of the classification process). Proper selection of network parameters, such as the number of convolutional layers, filter size, and the number of neurons in the fully connected layers, enables the convolutional network to detect important features and effectively classify sessions based on encoded URLs. The smallest error on the test string obtained by the network was 26.9% (Fig. 3.6).

In Fig. 3.7 the values of filters of the first layer of trained CNN are shown.

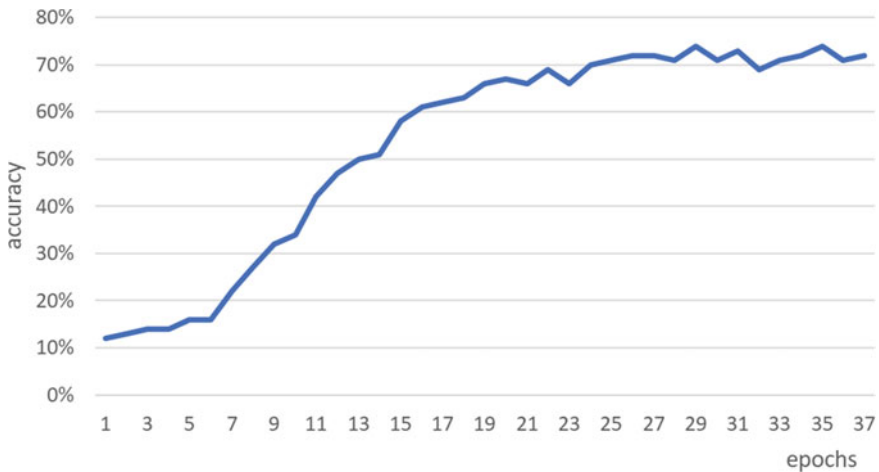


Fig. 3.6 The course of the learning process (quality of operation in relation to the number of epochs)

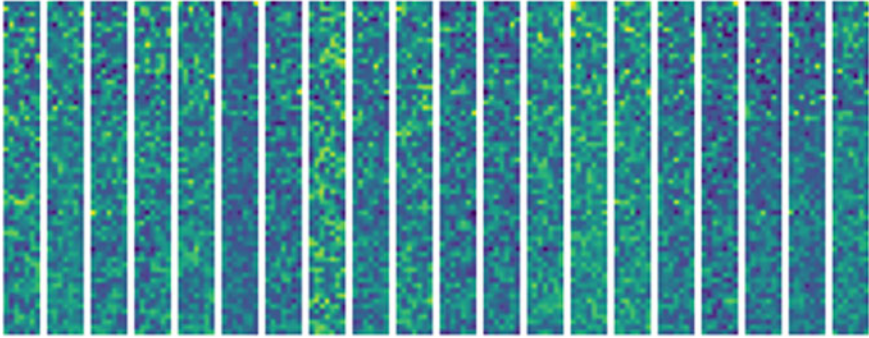


Fig. 3.7 Visualization of CNN's first layer filters for user classification (light colors mean higher values)

3.3 User Profiling Using Recurrent-Convolutional Networks

This section focuses on advanced neural network models, such as LSTM (long short-term memory) and GRU (gated recurrent unit), which are particularly useful in sequential data analysis. These models have gained popularity in the context of natural language processing, due to their ability to efficiently store and use long-term dependencies in sequential data. LSTM and GRU networks are extensions of classical Recurrent Neural Networks (RNNs) [30]. Unlike standard RNNs, which have difficulty storing long-range information in sequential data, LSTM and GRU introduce additional control mechanisms that allow for efficient analysis and use of long-term dependencies. An LSTM network introduces a special memory cell structure that can store and update information depending on the input and state of the previous iteration. So an LSTM network is able to retain important information for a long time and avoid the phenomenon of vanishing or exploding gradient, which often occurs in standard RNNs. A GRU network introduces a simplified memory cell structure, which consists of only two gates: a reset gate and an update gate. These gates allow a GRU network to control what information is stored and updated in the memory cell. Both types of networks, LSTM and GRU, have their unique features and applications, and the choice between them depends on the specific problem and data. Further information on these models, their structure, operation, and practical applications will be provided later in this section.

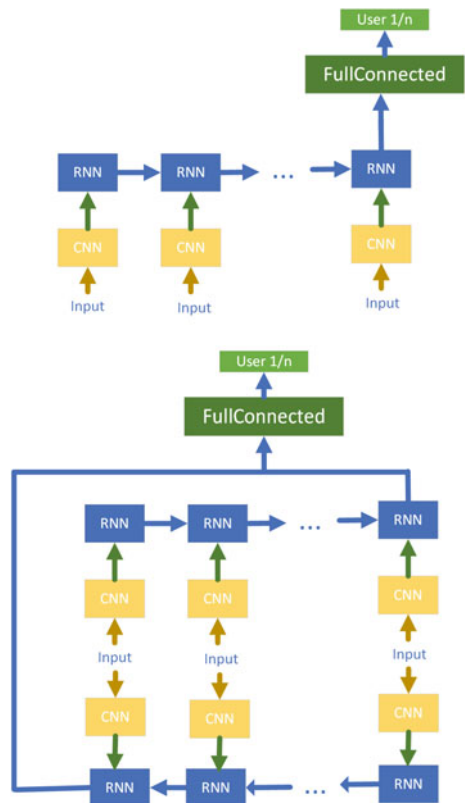
User profiling can also be performed with the help of convolutional-recurrent networks. Such structures may seem to be a natural tool to use in the problem of profiling users of computer networks. It should be remembered that in the case of convolutional networks, the size of the input vector had to be arbitrarily determined, thus, depending on the number of URL requests, information was lost (the input signal was shortened) if a specified size was exceeded, or the input vectors had to be filled with 0 value in the absence of sufficient data. This difficulty does not occur in

recurrent networks. In this section, unidirectional recurrent networks LSTM (long short-term memory) and GRU (gated recurrent units) [31, 32], as well as bidirectional recurrent networks BiLSTM (bidirectional long short-term memory) and BiGRU (bidirectional gated recurrent units) were tested. For birecurrent structures, we use the same two cells—LSTM or GRU simultaneously in a single network model. The general concept of operation of the networks used in the paper is shown in Fig. 3.8.

The variant of a recurrent network used in this section is commonly referred to as “many-to-one”. [33] In the presented case, it is not important for the model how many addresses a given session consists of, since this number naturally affects the number of calculation calls in the recurrent cell.

Figure 3.9 depicts the progression of outputs when feeding successive URL addresses into a recurrent neural network. The output has been limited to 5 users to make the changes more visible on the graph. Thus, many URLs in a given session are used to indicate a single user ID encoded in a vector of “1 of n ” type. Analogically to the case described in the previous section, a single URL was recorded in a $52 \times 70 + 2$ matrix, where 52 represents the maximum number of letters in a single URL, 70 represents the number of letters in the adopted alphabet, and 2 represents the time

Fig. 3.8 Diagram of recurrent networks used in user profiling problem



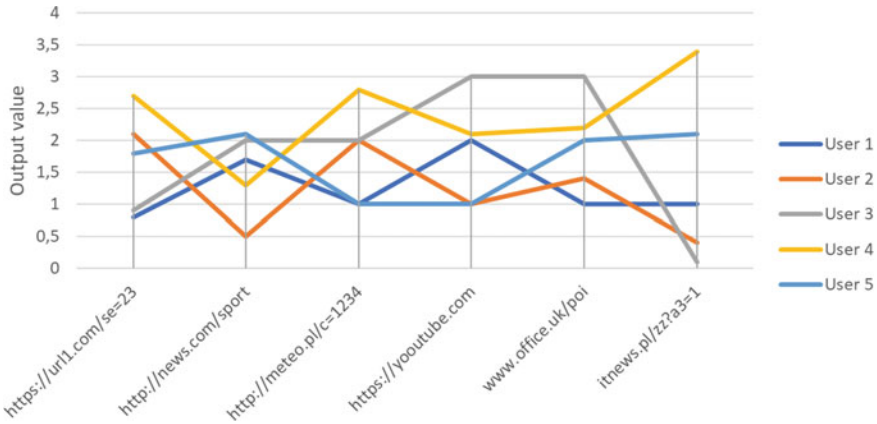


Fig. 3.9 Presentation of recurrent network outputs when calling subsequent addresses from the sequence

Table 3.4 A recurrent network model for network user profiling, variant 1

Layer No.	Layer description
1	Input with a size of: $52 \times 70 + 2 \times 1$
2	1D convolution with ReLU activation, 128 feature maps, 1D filter, width 5, step 1
3	1D convolution with ReLU activation, 256 feature maps, 1D filter, width 3, step 1
4	MaxPooling 1D filter, width 6, step 3
5	Recurrent cell, inner size 384
6	Output 1 of 68

of day like in the previous studies. As a result of numerous experiments in which various models of recurrent networks were subjected to the training process, the best results were obtained for the structure presented in Table 3.4.

Tables 3.6 and 3.7 present training times and obtained results of simulations for different types of recurrent networks.

It is easy to see that the best result was obtained for the BiGRU network. However, comparing this solution with the one presented in the section on convolutional networks, it should be noted that a minimally better result was obtained for the CNN network. In addition, the training time of this solution is longer in each of the variants of the recurrent network. Data contained in Tables 3.6 and 3.7 are averaged after a 20-fold repetition of the simulation.

Table 3.5 A recurrent network model for network user profiling, variant 2

Layer No.	Layer description
1	Input with a size of: $52 \times 70 + 2 \times 1$
2	1D convolution with ReLU activation, 64 feature maps, 1D filter, width 5 step 1
3	1D convolution with ReLU activation, 128 feature maps, 1D filter, width 3 step 1
4	1D convolution with ReLU activation, 256 feature maps, 1D filter, width 2 step 1
5	MaxPooling 1D filter, width 6, step 2
6	Recurrent cell, inner size 384
7	Output 1 of 68

Table 3.6 Training times in one epoch for all neural networks with two variants of input convolutional layers

Type	Time [s/epoch]	
	Variant 1	Variant 2
LSTM	132	186
BiLSTM	211	231
GRU	101	128
BiGRU	198	229

Table 3.7 Result of operations of recurrent networks

Cell version	Variant 1				Variant 2			
	Qualification accuracy (%)	Precision	Recall	F1	Qualification accuracy (%)	Precision	Recall	F1
LSTM	70.23	0.70	0.61	0.69	71.80	0.71	0.61	0.70
Bi LSTM	77.10	0.76	0.72	0.73	76.80	0.70	0.62	0.68
GRU	72.42	0.71	0.70	0.70	73.40	0.70	0.70	0.68
BiGRU	76.60	0.76	0.71	0.72	76.41	0.72	0.72	0.71

3.4 Detection of Network Traffic Anomalies Using OC-NN

One-class classifier (OCC) networks are a special type of models designed to identify and recognize only one particular class or category in a set of many different classes of data. Training of a one-class network is based on only one class of data, which is considered to be the expected one. The model tries to learn the representation of the selected class and creates a decision boundary that contains most of the data belonging to it.

In previously conducted simulations, the proposed solutions were aimed at assigning a user ID to input data. The solution proposed here has a different purpose, which is detecting anomalies and therefore, the task for the neural network is different. The goal now is to verify that the input data set matches the provided user ID. In other words, the task of the model is not to determine who the input data belongs to, but to verify if the provided input data is correct for a particular user. The first solution to this problem was implemented using the previously described convolutional network. An example of such a network is shown in Fig. 3.10. Its structure is analogous to the one previously presented for the classification of computer network users, however, it was modified by: adding a binary network output and user ID encoded with the “one-hot” method to the input signal.

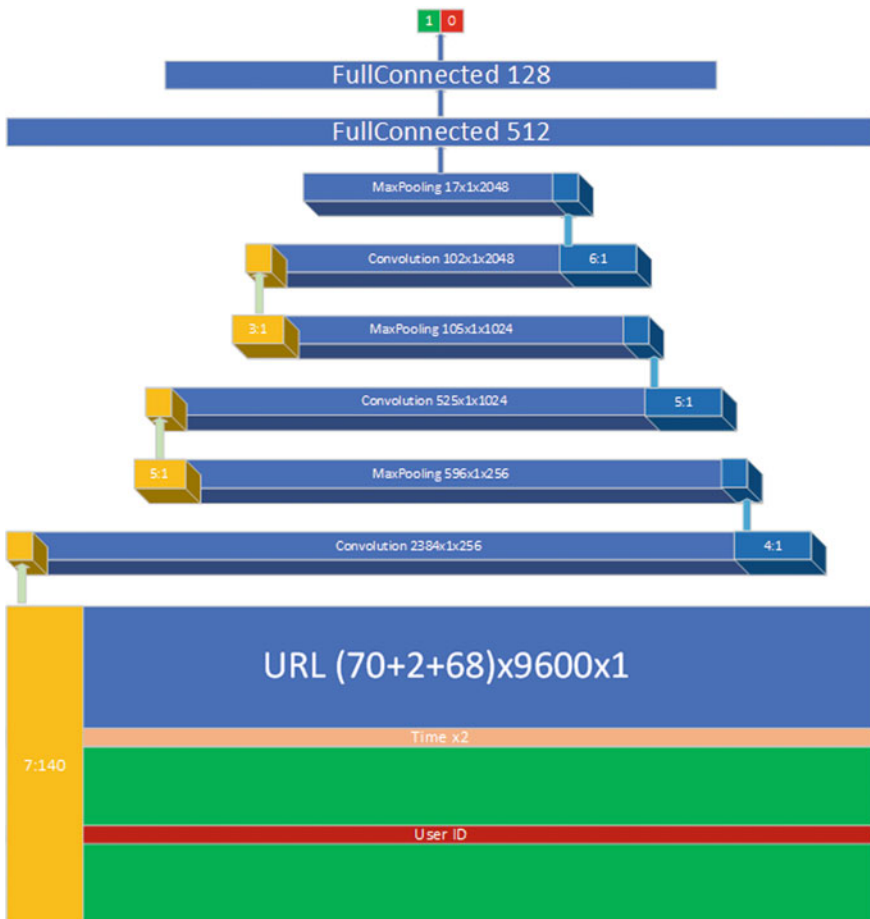


Fig. 3.10 CNN network for detecting anomalies

Figure 3.10 shows a comparison of the proposed network model with the classic OC-NN solution. In this case, the OC-NN network was implemented using multiple autoencoders, where the input data is marked with an identifier pointing to the user who generated it. A reference OC-NN network is used only for the analysis of samples of one class. The main task of the encoder is to extract features that enable classification, and for this purpose, a convolutional network structure is used.

An autoencoder is a type of neural network used to detect and reproduce important features of input data. It consists of two main parts: encoder and decoder. The encoder converts the input vectors into a smaller representation that contains the most important (compressed) information. The decoder then reproduces the input based on this shortcut. The main goal of an autoencoder is to minimize the reconstruction error, that is, to reproduce the input data as accurately as possible at the output.

The autoencoder is trained on the input data, reconstruction is compared with the original data and network weights are updated in order to minimize the differences between them. Autoencoders are used to reduce the dimensionality of data, extract relevant features, denoise data (DEA—denoising autoencoder) and generate new data examples. They are particularly useful for high-dimensional data, where there is a need to find a low-dimensional representation of data while preserving relevant information. In the context of the solution described earlier, autoencoders are used to prepare classifiers within one-class neural networks (OC-NN). Each encoder is trained as a classifier for a specific class, and then a collection of trained autoencoders is used to assess whether a given session belongs to a given user.

In the case of the autoencoder structure used in the described solution, some differences can be observed compared to traditional autoencoder structures. In this case, the network relies on a natural language processing (NLP) approach [34], which is reflected in the encoder, which generates a hash based on the URL session. Autoencoder training data is created based on a set of URLs that are treated like words in a sentence. No separator between addresses is used, and the URL itself is treated as a word in a sentence. This approach makes it possible to treat a URL session as a unit on which natural language processing techniques can be applied. The structure of an autoencoder used for texts is different from the structure of an autoencoder used for images [35]. In this case, a network inspired by [36] and adapted to this task was used. It is worth mentioning the output of the encoder described earlier, with a size of 128×64 . Initially, attempts were made to create an autoencoder using a modified convolutional network with a U-Net [37] structure, but without links between feature maps of the same size. Unfortunately, this solution did not bring satisfactory results in the training process. The final solution used a “sub-pixel convolution” layer in one dimension, based on the technique described in [38]. The task of this layer is to expand the size of the layer at the expense of its height. This technique, combined with 1D convolutional networks, proved effective in generating spatial features in one dimension, which was essential for this task. A presentation of this task is shown in Fig. 3.11.

The chosen structure of the autoencoder turned out to be the optimal approach for the purpose of this task. However, it is worth noting that the size of the middle layers of the autoencoder, which contain semantic hashing of URL sessions, is relevant.

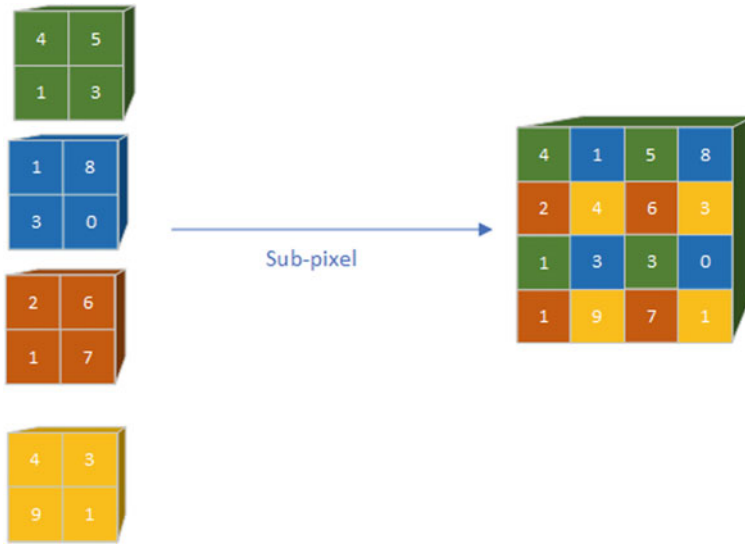


Fig. 3.11 Processing in the “sub-pixel convolution” layer

The wrong size of these layers can lead to poor text mapping (poor autoencoder quality) or difficulties while training the discriminator, even if the mapping is of very good quality. In the case of the tested solution, the dimension of the middle layer of the autoencoder was 125×64 , obtained from an input matrix of 72×4092 . The structure of this autoencoder is shown in Fig. 3.12. It is the result of experiments to find an optimal structure that provides an appropriate balance between the quality of mapping and the possibilities to train the discriminator.

In the design of the autoencoder, the BatchNormalization method was applied after each convolutional layer, except for the last, and 22nd layer. The ReLU activation function was used to activate the layers, except for the last one. It is worth noting that the structure of the autoencoder was the same for each user, which means that the same network configuration was applied regardless of the user. This uniform structure of the autoencoder allowed for effective comparison and analysis of results for different users (Table 3.8).

Trained autoencoders were used to build an OC-NN model. In both solutions shown in Fig. 3.13, it is important to create a classifier that assesses whether the specified network traffic belongs to the selected user. The previous network-based solution was to assign input data to a specific user, while the current OC-NN solution focuses on anomaly detection based on the user ID.

Many simulations were performed to find the optimal network configuration. After analyzing the results, the optimal network structure was determined—it is presented in Table 3.9. This optimal network configuration includes an appropriate number of convolutional layers, batch normalization layers, activation functions, and other parameters that contributed to the best results in the problem under study. Table 3.9

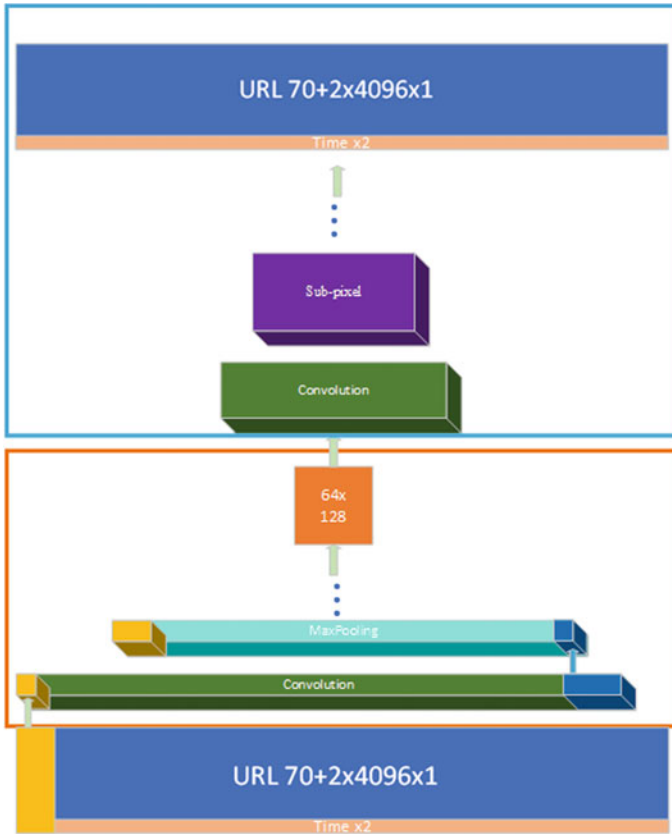


Fig. 3.12 Structure of the autoencoder for URL session

provides detailed information on this optimal network design, thanks to which we can understand the network configuration used in the experiments.

Information fed to the network input during the training and testing process consisted of two signals: URLs and a user ID, encoded as a “1 of n” vector. The task was to determine whether the input values are correlated, that is, whether the encoded network traffic corresponds to the user ID. After assessing the quality of the trained structure, it was found that its efficiency was only 62.1%, which in the case of binary classification means a result which is only minimally greater than the random value. To improve the results, an adaptation of a one-class neural network (OC-NN) solution proposed in [39] was applied. A proprietary modification was introduced. It consisted in adding a user ID to input data and using one classifier that uses data from different encoders [40]. This solution combines the idea of an OC-NN with the networks used for classification. The network structure is shown in Fig. 3.13. OC-NN operation consists in preparing a separate classifier for each class, which in the training process takes the form of an autoencoder. At the end of the training

Table 3.8 Overview of the autoencoder layers

Layer No.	Layer type
1	Convolutional output (72, 4096)
2	MaxPooling output (72, 2048)
3	Convolutional output (256, 2048)
4	MaxPooling output (256, 1024)
5	Convolutional output (256, 1024)
6	MaxPooling output (256, 512)
7	Convolutional output (512, 512)
8	MaxPooling output (512, 256)
9	Convolutional output (512, 256)
10	MaxPooling output (512, 128)— discriminator input
11	UpPooling output (128, 128)
12	Convolutional output (256, 128)
13	UpPooling output (128, 256)
14	Convolutional output (512, 256)
15	UpPooling output (256, 512)
16	Convolutional output (512, 512)
17	UpPooling output (256, 1024)
18	Convolutional output (256, 1024)
19	UpPooling output (128, 2048)
20	Convolutional output (256, 2048)
21	UpPooling output (128, 4096)
22	Convolutional output (72, 4096) activation function: Sigmoid

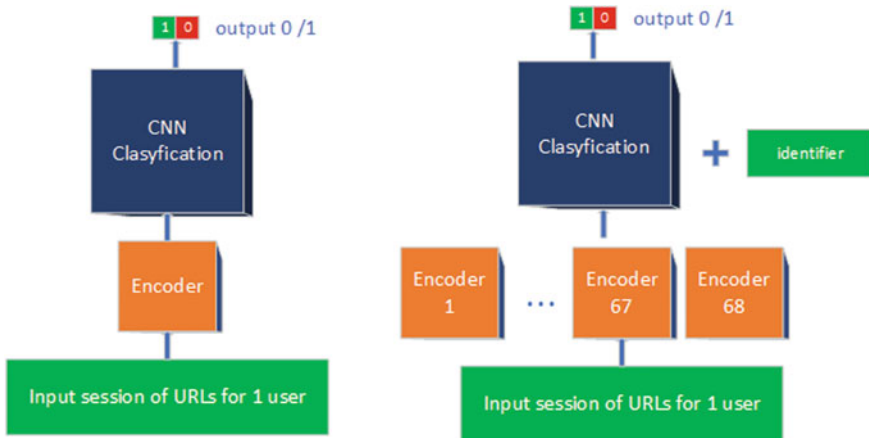


Fig. 3.13 Scheme of operation of OC-NN structures described in the literature and the scheme presented in this book

Table 3.9 Convolutional network of the OCC-NN type for detecting anomalies

Layer No.	Layer description
1	Convolution 128 FM, filter size 7, step 3, ReLU activation
2	MaxPooling filter size 3, step 3
3	Convolution 256 FM, filter size 5, step 2, ReLU activation function
4	MaxPooling filter size 3, step 3
5	Convolution 256 FM, filter size 3, step 1, ReLU activation function
6	MaxPooling filter size 3, step 3
7	Fully Connected 512 + Dropout, ReLU activation function
8	Fully Connected 256 + Dropout, ReLU activation function
9	Output 2, softmax activation function

process, a set of trained autoencoders was obtained. Then, by “splitting” them as shown in Fig. 3.11, a set of models was obtained that can be used to compress the information. The encoded URL session, along with the ID, was then fed into the input of the discriminator, which assesses whether the session belongs to a given user. In this case, the discriminator had the structure of a convolutional network, used to classify text [23, 41]. The network structure is shown in Fig. 3.5.

As part of tests, the presented method was modified by creating one autoencoder for all users. Compared to a traditional CNN network, this solution brought better results. However, it did not provide a sufficient solution to the problem, since the anomaly detection error was greater than in the case of dedicated autoencoders for individual users. The training process of the autoencoder was performed using the binary cross-entropy loss function. This feature is commonly used in unsupervised training tasks, such as training autoencoders, where the goal is to reproduce input data as faithfully as possible at the output. By minimising the value of binary cross-entropy, the autoencoder tries to reduce the error of input data reconstruction, which contributes to learning effective representation of features. The conclusion is that modification of the presented method using a single autoencoder for all users brought some benefits, but there is still a need for further adjustments to improve the performance while detecting anomalies in user behavior.

The Sørensen similarity coefficient or Dice coefficient, is a measure of similarity between two samples or collections. It is widely used in fields such as image processing, natural language processing, bioinformatics and many more.

The quality assessment of the trained autoencoder model was performed using the Sørensen similarity coefficient. The coefficient is defined as the ratio of the double number of elements common for two sets (A and B) and the sum of the number of elements in both sets. It can be expressed mathematically as:

$$QS = \frac{2 * C}{A + B} \quad (3.2)$$

where A and B are the numbers of all points of images A and B , respectively, and C is the number of points common for both images. The value of the Sørensen similarity coefficient ranges from 0 to 1, where 0 means no similarity between sets and 1 means full overlapping of sets. The higher the value of the coefficient, the greater the similarity between sets. In practice, the Sørensen similarity coefficient is often used to compare two sets or samples to assess their degree of similarity or overlapping.

The discriminator training was performed using the “cross entropy with softmax” loss function. In the case of OC-NN network training, the training factor is defined as follows:

- For the first 10 epochs the training factor was 0.001,
- For the next 15 epochs the training factor was 0.0001,
- For the next 20 epochs the training factor was 0.00005,
- The maximum number of training epochs was 580.

When training autoencoders, the best results were obtained using the following training factor values for each user:

- For the first 10 epochs the training factor was 0.001,
- For the next 300 epochs the training factor was 0.0001.

In both cases of training, that is, for both the autoencoder and the CNN network, the stochastic gradient descent (SGD) algorithm with a momentum value of 0.9 was used. This gradient optimisation algorithm is widely used in training of neural networks [42].

In order to ensure a balance in the sample representation for each class, fifty-five users with a similar number of logs were selected for the study. On average, there were 23,860 sessions per user, which were called positive sessions. In order to assess the quality of the developed algorithms, negative vectors were introduced into the training and testing sets. Negative vectors are data in which the label (class) that denotes the user has been changed. The process of generating the testing and training set included the following steps:

1. network traffic data was collected and divided into sessions for individual users, as previously described.
2. An equal number of negative and positive sessions were generated for each user. To achieve this, a user ID was assigned to data from sessions of other, randomly selected users.
3. Sessions were saved with a Boolean value that indicates if the session is false or true.

Table 3.10 contains a summary of the experiments results. It was confirmed that the best results were achieved using a network with dedicated autoencoders [43] for individual users.

Table 3.10 Results of simulation tests for anomaly detection in network traffic

System	Qualification accuracy (%)	Precision	Recall	F1
CNN OC network	62.1	0.62	0.51	0.51
OC 1 × autoencoder network	69.2	0.63	0.61	0.67
OC 68 × autoencoder network	83.1	0.83	0.81	0.79

When analyzing the obtained results, attention should also be paid to the quality of the operation of autoencoders. In the proposed structure, the best mapping efficiency of encoded URLs was 92%. However, there were exceptions for six users, where mapping efficiency for the testing set was 71.1–73.7%. The average quality of autoencoders for all users was 83.1%. The conducted experiments clearly show that methods based on convolutional networks are very effective in tasks involving a creation of user profiles and detecting anomalies in computer networks. It should be emphasised here that even most proficient computer network administrators will not be able to analyze such large data sets and achieve such fast response times without tools based on AI techniques.

Chapter 4

Convolutional Glial Networks



Glial cells, also known as glia, are cells found in the brain and nervous system alongside neurons. They are necessary for the proper functioning of the nervous system and perform a variety of supporting and protective functions. Glial cells make up about 90% of all cells of the brain. There are several types of these cells depending on their functions, e.g., astrocytes, oligodendrocytes and microglia. Astrocytes are the largest group of glial cells. They perform many functions, such as maintaining chemical and ion homeostasis in the brain, supporting neuronal metabolism, participating in the formation and maintenance of synaptic connections, and delivering nutrients and oxygen to neurons. Oligodendrocytes are responsible for the production of myelin in the central nervous system, while in the peripheral nervous system this role is played by Schwann cells. Oligodendrocytes are responsible for the production of myelin sheaths, which surround the axons of neurons in the central nervous system. Myelin sheaths provide isolation and accelerate the conduction of nerve impulses. Microglia perform an immune function in the brain. They are the first line of defence in the event of damage to nerve tissue and are involved in inflammatory processes and the removal of dead cells and pathogens. Glial cells also perform supportive functions such as delivering nutrients and oxygen to neurons, maintaining ion and pH balance, removing toxins and metabolic waste, and providing structural support to neurons. Abnormal functioning of glial cells can contribute to the development of various neurological diseases, such as multiple sclerosis, Alzheimer's disease, Parkinson's disease and brain tumours. Perhaps the most important finding in the field of brain research is that glial cells are involved in neuronal communication. Astrocytes, for example, can secrete chemicals called gliotransmitters that influence the functioning of neurons and the transmission of signals in the brain. Glial cells play an important role in neuronal development. During embryogenesis, glial cells are involved in the formation of brain structures, the migration of neurons, the formation of synaptic connections and the formation of myelin sheaths. Glial cells are also associated with synaptic plasticity, i.e., the ability of synapses to become stronger or weaker. Studies suggest that astrocytes can affect synaptic processes such as long-term potentiation

(LTP) and long-term depression (LTD). They also play an important role in the repair and regeneration of the brain after damage. Oligodendrocytes can repair damaged myelin sheaths, and astrocytes can form glial scars that protect damaged areas of the brain. Continuous broadening of knowledge on glial cells opens up new perspectives in the field of therapy and treatment of neurological diseases. For example, research on stem cells indicates that they have a potential to repair brain damage by differentiating into glial cells or neurons. Investigations of glial cells are ongoing, and the exact functions of these cells in the brain are still being studied. However, we already know that they are extremely important for the nervous system to function properly and that they have a significant impact on cognitive processes and condition of the brain.

The main subject of investigations described in this section is the assessment of the potential of applying the latest findings on the structure and function of the human brain to optimize the design of artificial neural networks. In numerous recent medical publications, we find reports on the influence of glial cells on the formation of connections between neurons in the brain and the transmission of information. An important issue of the last two years is research on the use of glial cells in oncology, cardiology and dentistry. Polish doctors from Wrocław are pioneers in these fields. Based on their knowledge of glial cells, they reconstructed a severed spinal cord [44, 45]. In his book “The Other Brain” R. Douglas explains the importance of these cells for thought processes and addresses important issues such as:

the results of recent studies on mutant mice with a disturbed sleep cycle which made many scientists aware that the other brain is not just a shadow of the neuronal brain or a slave which satisfies its needs- the other brain can control the neuronal brain. According to Douglas, astrocytes coordinate the work of clusters of neurons that cause neurons to rhythmically produce impulses, like a conductor coordinating a group of musicians in an orchestra playing a piece of music.

The analysis of these issues is a key aspect of studies described in this section, which focus on the search for synergies between the latest developments in the field of neurobiology and the design of advanced models of artificial neural networks. Based on the collected information, an experiment was conducted to create a new neural network architecture with an ability to supervise the process of building and training convolutional networks. This approach refers to existing research on the human brain, which indicates that glial cells protect neurons and are responsible for establishing connections between them. With this in mind, new structures of neural networks were developed that are able to control the activity of connections between neurons by adjusting the values of weights, which leads to the elimination of unnecessary connections and thus unnecessary neurons or groups of neurons. This section presents an adaptation of existing training algorithms to this new architecture (Fig. 4.1).

New concepts of neural network structures and their training techniques were verified in the context of ensuring the security of information systems, based on the analysis of network traffic. This process is very complex, especially due to the huge amount of data processed, which directly affects the calculation time. If a threat is detected, a quick response is required, so it is crucial that the neural network acts on

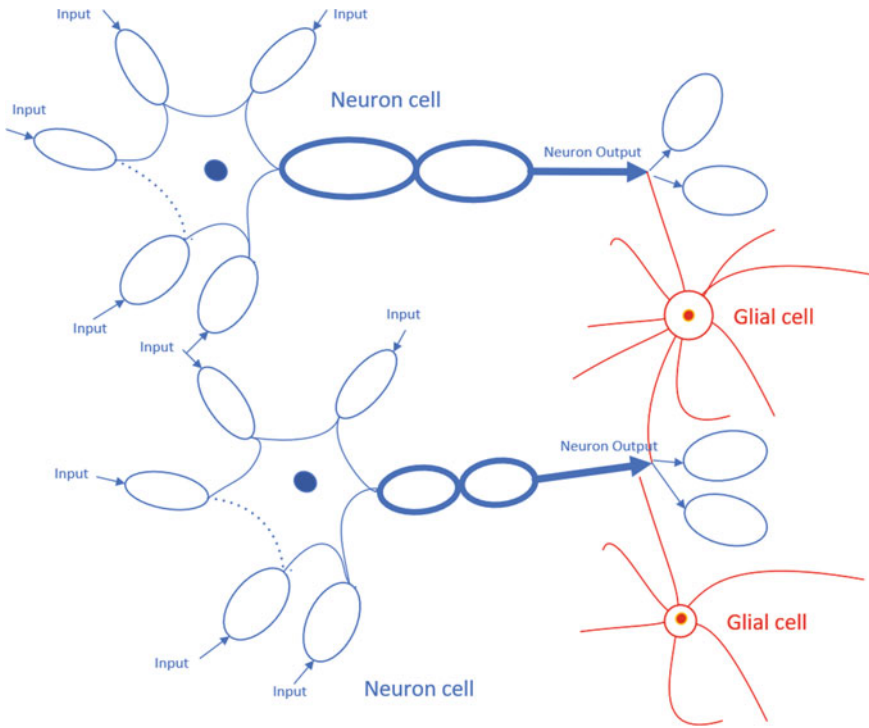


Fig. 4.1 Neurons and glial cells (illustrative drawing)

the given input signal in the shortest possible time. It is due to the need to optimize the network structure, since reducing its size results in a significant reduction in the number of calculations and time of response to the input signal. This section presents proprietary methods of optimizing neural network structures using the example of models of network structures based on glial cells, described in the previous sections.

4.1 Optimization of Convolutional Networks Using Glial Cells

Artificial neural networks constitute the basis of the most widely used algorithms for data analysis today. However, they are treated as “black boxes” with a specific role. The quality of their operation can be assessed on the basis of statistical measures and test samples. Although the way neural networks work is known as are the algorithms adapting structures to correct operation on specific data sets, the current state of knowledge does not allow us to formulate the rules of operation for processes embedded in the model. Methods of knowledge transfer between networks are still

unknown. This book contains an analysis of the processes accompanying the training and operation of convolutional networks, with focus on the interpretation of the knowledge stored in them using the author's model inspired by biological glial cells. A CIFAR-10 dataset described in the literature is used to demonstrate the capabilities of the proposed new network structures [46]. It is a popular dataset used in the fields of image recognition and machine learning. It consists of 60,000 RGB images with a size of 32×32 pixels, divided into 10 equal classes. Each class represents specific objects and scenes, such as cars, birds, dogs, cats, airplanes, etc. CIFAR-10 consists of 50,000 training images and 10,000 test images. It is widely used to evaluate and compare the performance of different machine learning algorithms in an image classification task. It is a valuable dataset applied in the study of various machine learning techniques, including neural networks, convolutional neural networks (CNN), and other image classification algorithms. Because of its popularity, there are many scientific publications and studies that rely on it as a benchmark for evaluating the effectiveness of various methods in the field of image recognition. The best results in terms of classification of this set of information using the "standard" convolutional network were achieved in the studies [47, 48]. Literature present network types [49, 50] that allow us to achieve better final results. However, in order to demonstrate the advantages of the proposed concept, a comparison of functionality will be performed on the basis of well-established structures presented in the literature [49]. Within the studies on glial cells in convolutional networks, three experiments were conducted to test the operation of structures with glial cells.

4.2 The Use of Glial Cells to Optimize the Structure of a Convolutional Network with a Known Structure on the Example of an Image Classification Task

Pruning is the process of reducing excess parameters in neural networks while maintaining the quality of their operation. It consists in removing unnecessary connections or sometimes entire neurons.

There are different methods of optimising (pruning) a network, but the general process can be described as follows:

1. A neural network is first trained on a given dataset.
2. Then, the significance of each weight in the network is calculated based on their impact on the cost function. It can be done, for example, by analyzing the backpropagation gradient.
3. Based on a certain threshold or criterion, weights or neurons are removed. It can be, for instance, a simple removal of all weights below a certain threshold.
4. After the neural network is pruned, it is often re-trained on the same dataset or on a reduced dataset in order to adjust the remaining weights and optimize its operation.

5. After such process the neural network model takes up less memory and requires less computing power to operate. It is extremely important when using such a structure in an environment with limited resources, such as mobile devices or embedded systems.

Techniques that were used to optimize the structures of dense multilayer neural networks do not work well in convolutional networks, due to the need to perform a very large number of calculations. Therefore a concept of using glial cells to simplify convolutional network structures at the level of aggregated feature maps was developed and will be presented below. In Fig. 4.2 these blocks are marked as “convolution blocks”. For this purpose, a convolutional network structure presented in [97] will be used, in which a correct classification result of 82% was obtained. A diagram of the original network [97] is shown in Fig. 4.2.

The proposed concept assumes that a convolutional glial network consists of two parts (two convolutional networks with a structure similar to that presented in [16]), a “glial driver” (Fig. 4.2, symbol Glial) and a CNN structure, each of which receives the same input signal during operation. The connection of the glial driver to the CNN is carried out by multiplying the output signal from a given convolution block by the output of the driver dedicated to this block.

Training such a structure required the development of an innovative concept of applying advanced optimisation techniques to reduce the structure of convolutional networks. The method is based on the use of existing deep neural network training mechanisms and advanced data analysis algorithms. The results of the simulations carried out as part of the experiment suggested that the proposed concept allows for an effective removal of unnecessary feature maps without adversely affecting the quality of network operation. In the case of the analyzed structure, the classification correctness remains at a high level of 84.1%, which proves the effectiveness of the method. The proposed training process required the development of dedicated strategies that involve alternating training of the CNN block and the “glial driver”. Optimization of the network structure is achieved by appropriate adjustment of activation functions such as the sigmoid function (values from the range $(0, 1)$) for the outputs of the glial network and the ReLU function for the other network elements.

An example diagram of removal of a single feature map is shown in Fig. 4.3. The filters used to create the map are highlighted in green. Blue indicates filters which use the same map.

The proposed solution requires a non-standard way of training the structure. The process is divided into two stages: first we train the convolutional module, and then the glial module. At the first stage, the glial module has blocked weights and does not participate in the training process. At the second stage, the convolutional module has blocked weights while the glial module is trained. During the training of the glial module, the outputs of the glial network are initially randomly selected. Each feature map has an equal chance of being turned off or on for each class. The training process covered 120 epochs. After the training procedure, the network structure was optimized by removing filters that create feature maps for which the values of the outputs from the glial module were below the specified threshold (in the experiment,

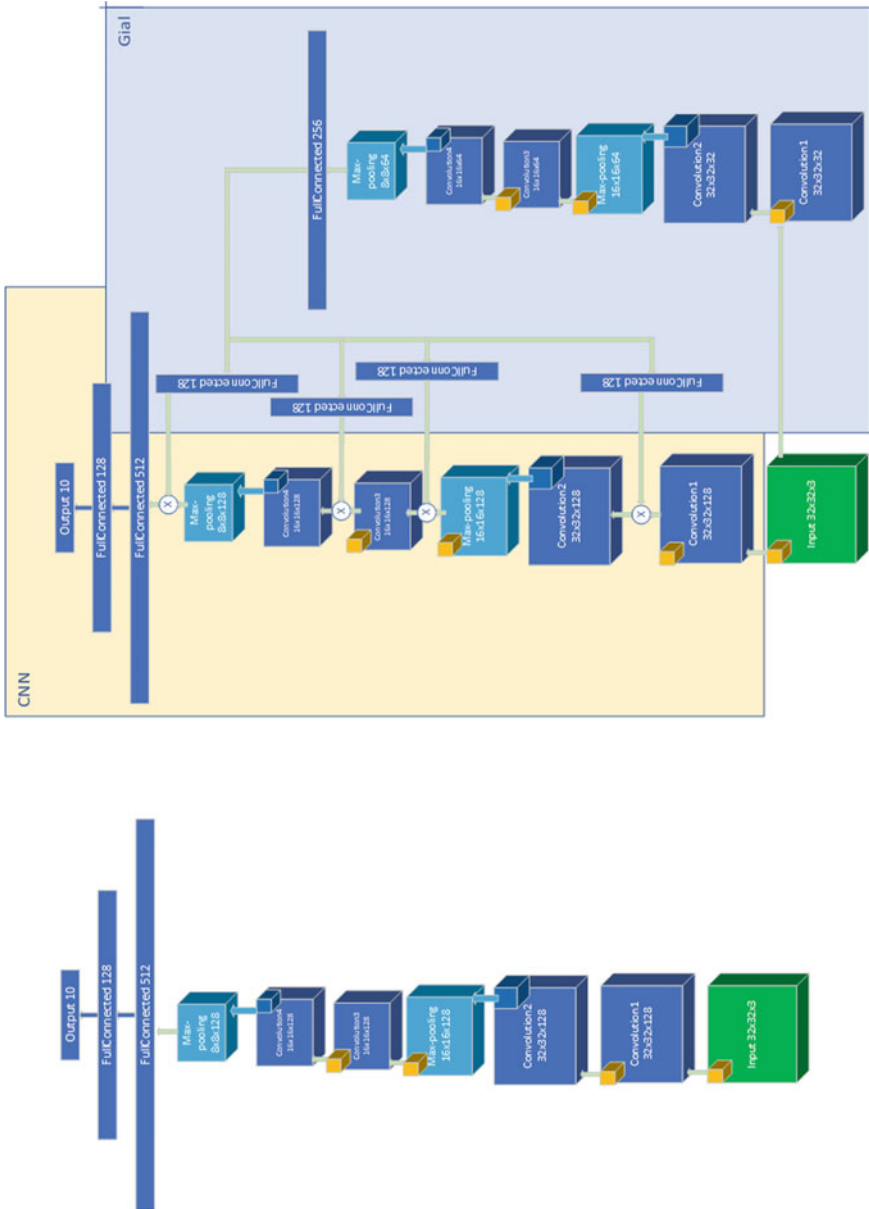


Fig. 4.2 Diagram of the original convolutional network proposed in [16] and propagation of the convolutional-glia network signal

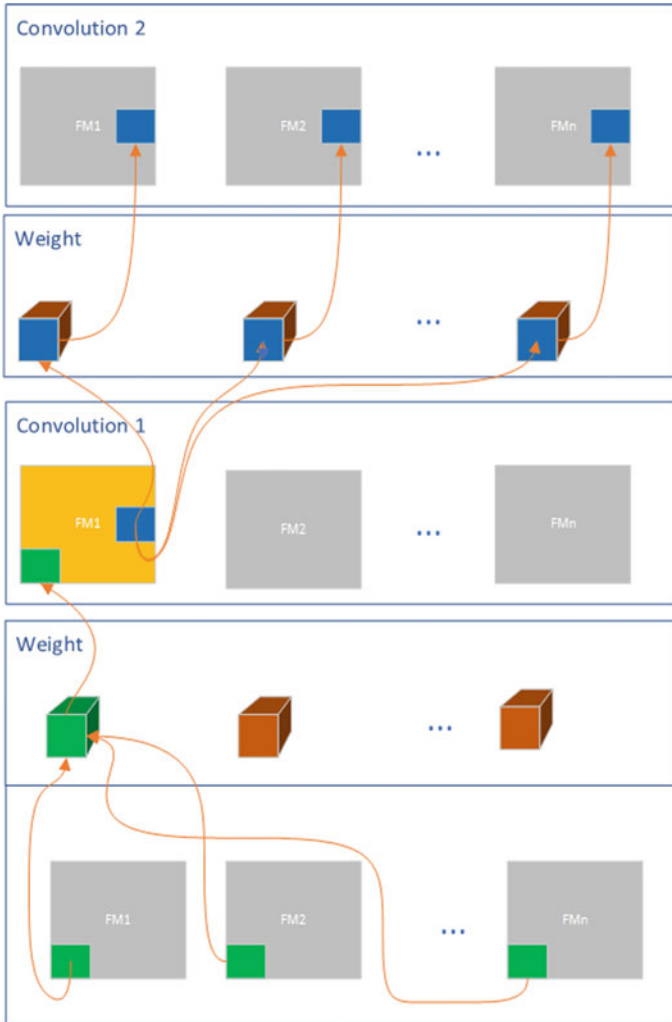


Fig. 4.3 Selection of filters and maps in a convolutional network

the threshold value of 0.3 for the sigmoid function was adopted). Switching off these feature maps did not adversely affect the classification result during the testing process. After the network structure was optimized, the weights in the glial module were randomly assigned again, and the training process was repeated. This approach allows for optimal selection of the convolutional network structure by switching off some feature maps based on the output values of the glial module. This process helps to simplify the network and can lead to better classification efficiency and effectiveness (Fig. 4.4).

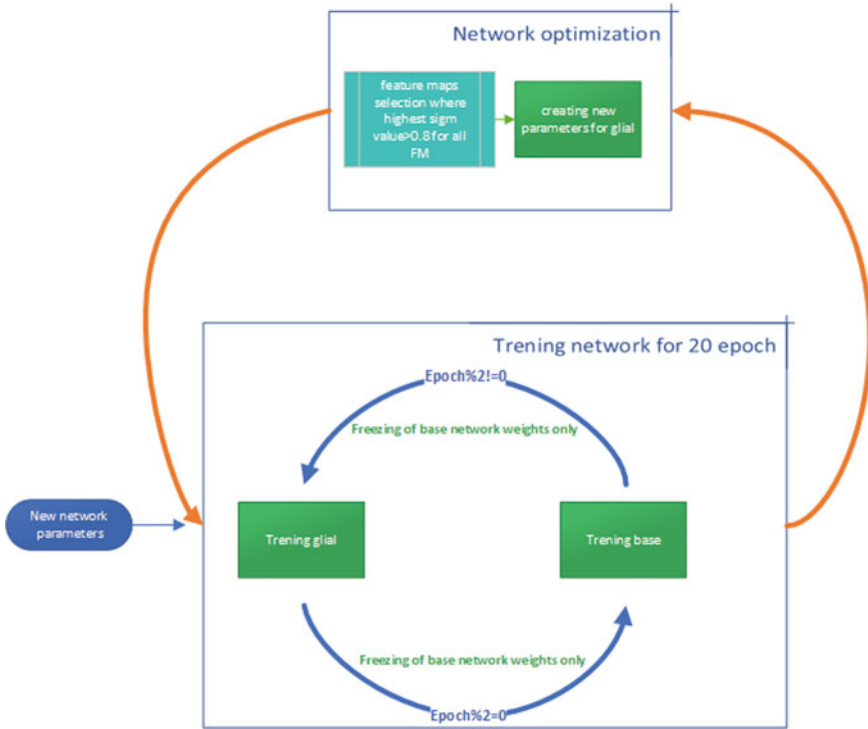


Fig. 4.4 The process of training the network with a glial structure

Thanks to the use of a glial network, it is possible to optimize the basic network to 82 feature maps, which accounts for 16% of all maps in the entire basic network structure.

4.3 The Use of Glial Cells to Optimize the Structure of the Convolutional Network When Solving Problems Using Previously Unknown Models

Overfitting is a phenomenon that occurs in machine learning when a neural network model is too closely adjusted to training data, which leads to poor generalisation for new, unknown data. In practice, this means that the model works correctly only for data coming from the training set, but unfortunately generates incorrect results for input information which does not belong to this set. It is often the result of excessive training of details and noise in training data that is not relevant to the overall pattern. Thus, overfitting leads to very high accuracy for training data, but low accuracy for data outside this set. This can occur when the model has too much capacity

(e.g., too many parameters in the neural network) relative to the available training data. The opposite of overfitting is underfitting. It occurs when a neural network is unable to adjust to the training data and cannot capture enough relevant patterns or dependencies present in the data. Symptoms of underfitting include poor accuracy for both training data and new test data. This phenomenon can occur when the model is too simple, that is, its capacity is too low for it to adjust to diverse patterns in the information contained in the training set. The goal for persons supervising the training process of the network is to find a balance between underfitting and overfitting, so that the model is sufficiently extensive, so that it can capture the dependencies in the data and at the same time is able to generalise its operation to new input information from outside the training set.

This section presents a technique that allows for an optimal selection of the structure (number of filters, layers and initial weights) of convolutional networks using glial cells. In the described methodology, the simplification of the structure consists in removing specific filters instead of entire convolutional blocks, unlike in the previously presented concept. The CIFAR-10 [51] base, which was described in the previous paragraph, was used to conduct the tests. The structure of the glial network was different from the one in the previous point, where the outputs of the glial driver were passed only to selected feature maps of the CNN network. To demonstrate the methodology of pruning the network, we began training the structure of the convolutional network along with an additional glial driver with a very large number of filters in the CNN network. To test how the glial network can optimize the size of the convolutional network, the proposed optimal output network was extended. A network diagram is shown in Fig. 4.5.

Table 4.1 shows data from the training process. Based on the experiment, it is possible to determine the optimal parameters of the designed convolutional network. The initial size of the structure was 256 feature maps in each convolutional block. The result of the first network training were 86.1% correctly classified samples from the test string. After 9 optimisation stages, the initial structure was reduced to 85, 95, 112, and 135 feature maps for subsequent network layers. For this architecture, the efficiency for the test string increased to 86.1%. The experiment was completed after 19 stages, when the quality decreased to 61.3%.

4.4 Using Glial Cells to Extract Knowledge from a CNN Network

Knowledge transfer between neural networks, also known as transfer learning, is a technique applied in machine learning that involves using knowledge and skills taught by one neural network to improve performance or shorten the training time of another structure with reference to a related task. The idea of knowledge transfer is that certain features and representations learnt by one neural network on one task can be useful for solving another, similar task. Instead of performing training from

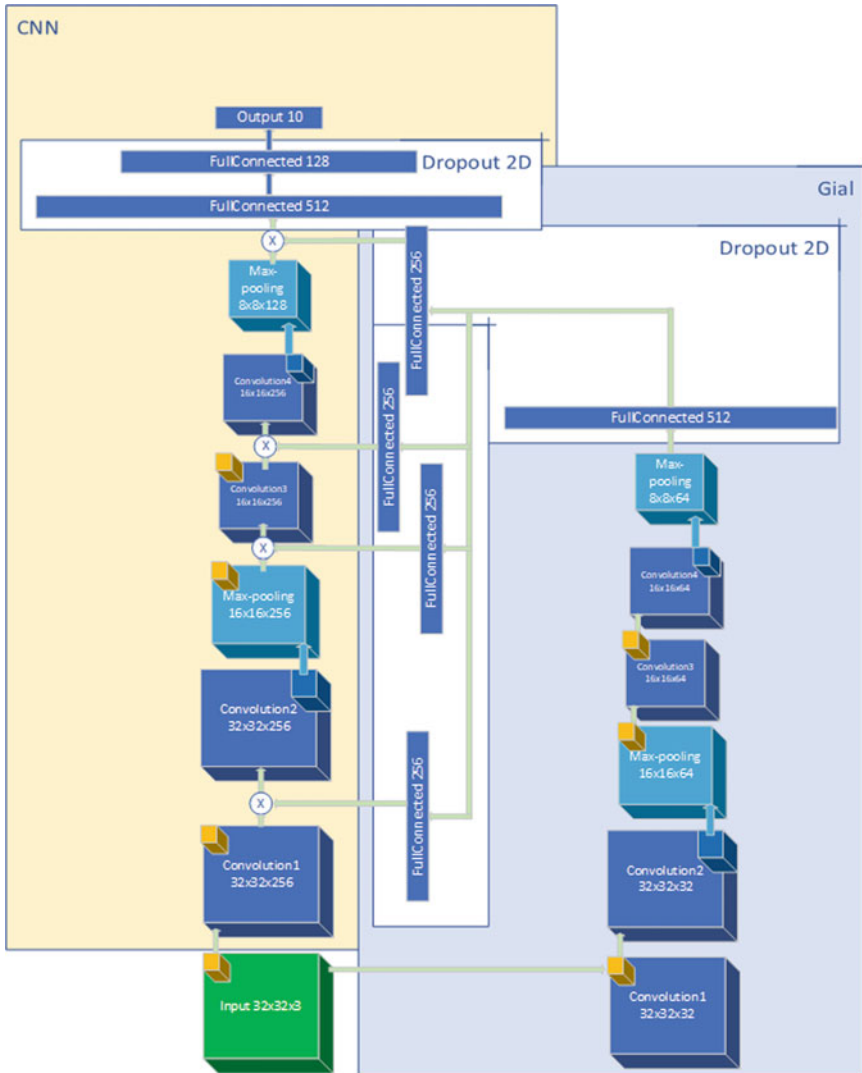


Fig. 4.5 The structure of the glial network used in the experiment

scratch on a new task, we can use already learnt weights, parameters and network structure as a starting point. The knowledge stored in the weights can include the extraction of features, patterns or data representations and its transfer can take place at different levels of neural networks, e.g., at the level of hidden layers, the output layer or even at the level of individual weights. There are different approaches to this issue, such as fine-tuning [52], where learnt weights are adjusted to a new task, or feature extraction, where already learnt features are used as a starting point for a new

Table 4.1 Process of learning CNN with network optimization using glial cells

Training	Size of layer 1	Size of layer 2	Size of layer 3	Size of layer 4	Efficiency [%]	Training time glia [s]	Training time CNN [s]	Switched off FM, layer 1	Switched off FM, layer 2	Switched off FM, layer 3	Switched off FM, layer 4
1	256	256	256	256	83.1	32	36	38	34	33	28
2	218	222	223	228	83.9	28	35	21	32	31	29
3	197	190	192	199	85.2	28	35	28	26	22	18
4	169	164	170	181	85.1	29	36	22	16	18	16
5	147	148	152	165	84.9	28	35	19	31	22	12
6	128	117	130	153	85.1	27	36	16	12	8	12
7	112	105	122	141	85.8	28	35	18	7	1	4
8	94	98	121	137	85.2	29	36	9	3	9	2
9	85	95	112	135	86.1	28	35	2	8	7	17
10	83	87	105	118	83.7	28	36	2	4	1	3
11	81	83	104	115	85.3	28	35	5	2	13	14
12	76	81	91	101	82.7	28	35	4	3	4	2
13	72	78	87	99	75.5	28	35	4	9	4	8
14	68	69	83	91	73.3	29	37	5	2	4	4
15	63	67	79	87	70.7	28	35	6	6	0	3
16	57	61	79	84	68.7	27	36	4	7	8	1
17	53	54	71	83	69.1	28	34	3	5	1	1
18	50	49	70	82	65.2	28	35	1	6	6	1
19	49	43	64	81	61.3	27	35	3	6	2	1

network. Knowledge transfer has many advantages, such as reducing the time and resources needed to train a new network, improving performance on smaller data sets, and the ability to generalise in case of entirely new topics. It is especially useful when limited training data resources are available or when the model needs to be quickly adapted to new tasks. It is a popular and effective technique in the field of machine learning, and many well-known neural network models were trained using knowledge transfer. In practice, knowledge transfer in convolutional networks can be applied in several ways:

1. **Fine-tuning:** It consists in adapting a trained convolutional network to a new task by continuing the training process, but at the same time freezing some layers and adjusting the weights only in chosen layers. The weights of these layers are initialised with the values of the trained network and then adjusted based on new training data.
2. **Transfer learning:** It is an approach in which learnt weights of a convolutional network are used as a feature extractor for a new network. Learnt convolutional layers are frozen, and only new layers are added and trained on a new task. This way we benefit from the network's acquired ability to detect image features, which often leads to better performance on a new task.
3. **Pretraining:** It refers to pre-training of a convolutional network on a large dataset (e.g., ImageNet) and saving the learnt weights. These weights are then used for initialisation in the convolutional network on the new task. Thanks to this, the network gets some knowledge of the image features, which speeds up the training process on a new task.

Knowledge transfer in convolutional networks has many advantages, such as reduced training time, better generalisation, better performance on smaller data sets, and ease of adaptation to new tasks. It is an important technique in the field of computer vision and is the basis for many advanced models and architectures, such as ResNet, VGG, and Inception, which were trained with it.

This section presents the possibility of using convolutional structures with a glial module in the field of knowledge transfer. The aim of the experiment presented below was to demonstrate that glial structures are able to indicate specific sets of filters in the convolutional network, which determine the decision to indicate a given class. Therefore, building a new network model based on specific sets of previously learnt filters should allow it to be initialized with specific knowledge (Fig. 4.6).

The structure of the convolutional network from the previous section was adapted for the purposes of the experiment. The presented model contains two types of functions: dropout 2D and 3D. These are regularisation techniques used in convolutional networks to prevent model overtraining. Both approaches are variants of the popular dropout method, which was introduced by Geoffrey Hinton and his team [53]. Dropout 2D (also known as spatial dropout) is used in convolutional layers and operates at the level of single feature maps. In each iteration of training, randomly selected units (neurons) are temporarily switched off (zeroed) with a certain probability. This action is aimed at eliminating the dependence between specific features and improving the overall ability of the model to generalise. Dropout 2D can be

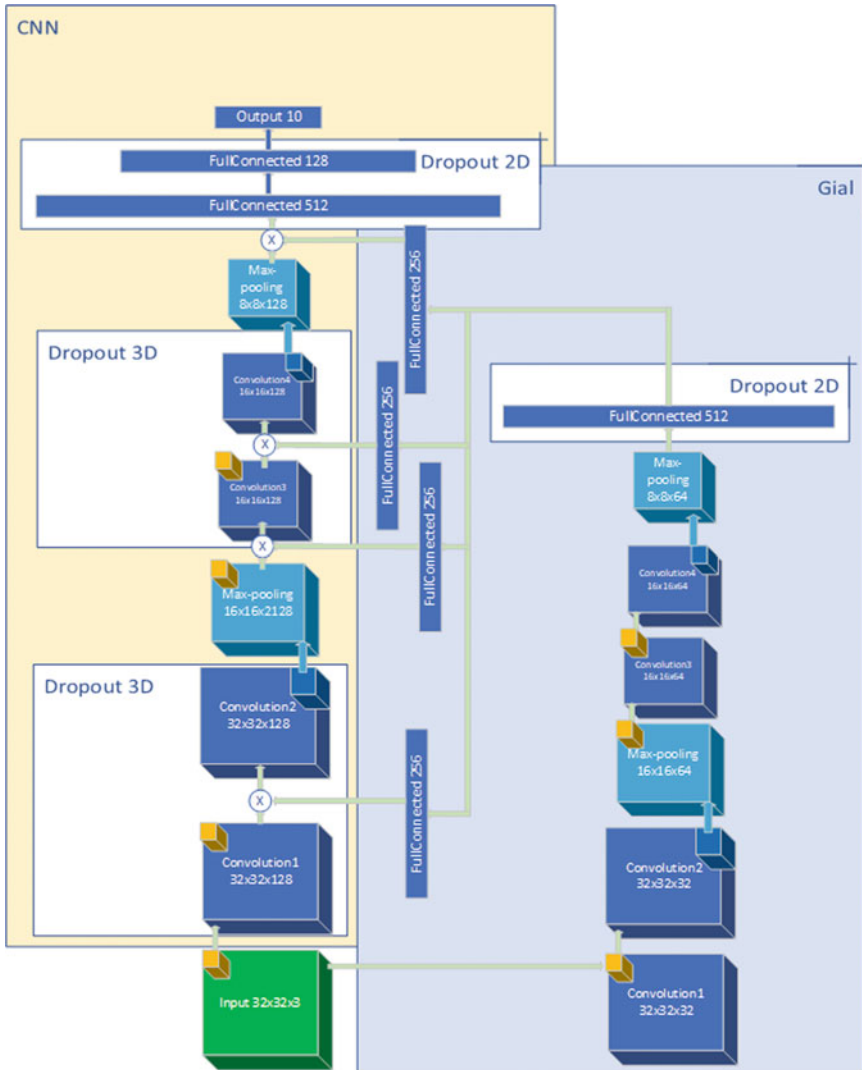


Fig. 4.6 Diagram of a glial network

applied after convolution or normalization layers or activation. Dropout 3D is a dropout extension to three-dimensional data, such as spatial data or sequential data in time. It is used in recurrent or convolutional layers where data has a third dimension (e.g. time or space). Dropout 3D helps to regularise the model, limiting excessive adaptation to specific areas of space or time. Both types (2D and 3D) are effective regularisation techniques in convolutional networks. They help to reduce excessive dependence between features, reducing model overtraining and improving its ability to generalise to new data.

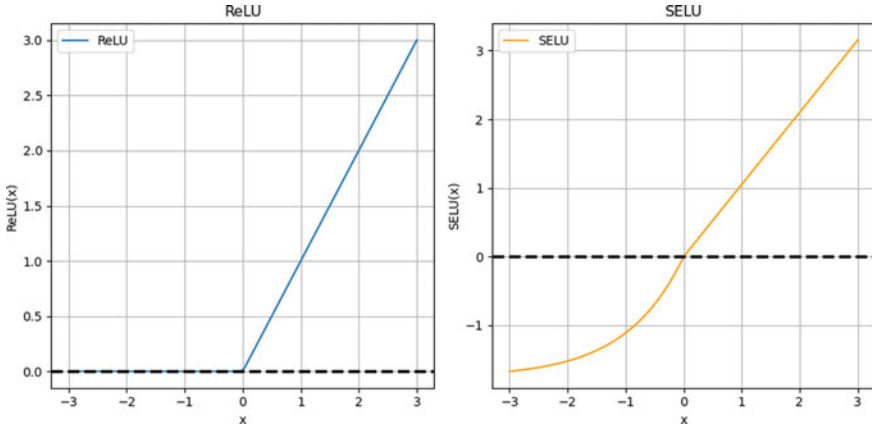


Fig. 4.7 Examples of ReLU and SELU activation functions

In order to demonstrate the knowledge transfer capabilities of glial networks, as in Sect. 4.2, a two-phase model training was performed. In the first phase, only the convolutional network parameters (without the glial network) were trained. In the second phase, only the glial network cells were trained. In order to optimize the training process, the value of the training factor was changed geometrically every 50 epochs (0.01; 0.001; 0.0001; 0.00001; 0.000001), and the momentum parameter was set at 0.9 at each stage of the simulation. A diagram of the training process is shown in Fig. 4.8. The best results were obtained by changing the activation function from ReLU to SELU [103], which is defined by Formula (2.31)

$$selu(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases} \quad (4.1)$$

SELU (scaled exponential linear unit) and ReLU (rectified linear unit) (Fig. 4.7) are two popular activation functions used in neural networks. ReLU has an output equal to zero for all negative input values (0 for $x < 0$), while SELU takes values from -1 to $(+)$ infinity.

Thanks to this SELU can generate both positive and negative values. The SELU function has a built-in normalisation property that promotes faster convergence of the neural network. Thanks to this, even without the use of normalization techniques such as Batch Normalization, SELU-based networks can effectively cope with the problem of vanishing and exploding gradient. In addition, SELU is a self-normalising activation function, which means that its output has a stable mean and variance even in deep neural networks. This can help to avoid the problem of vanishing gradient. The SELU function is more stable than the ReLU function, especially for large input values. ReLU in extreme cases can “generate” so-called “dead neurons”, while SELU provides greater variety and flexibility of outputs.

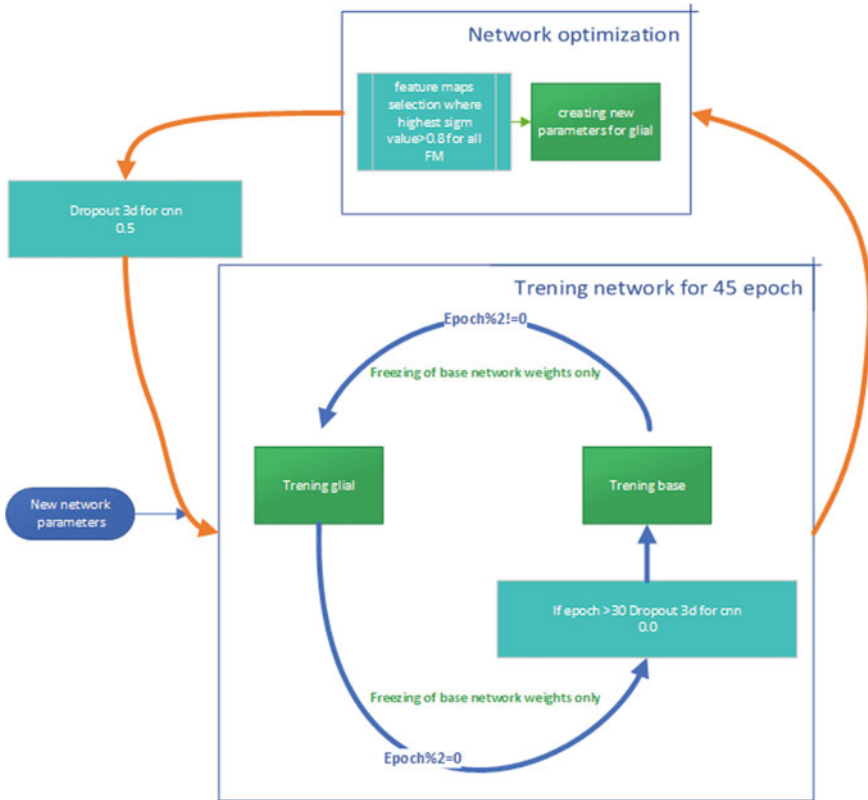


Fig. 4.8 The process of training the network with a glial structure

This function uses two additional parameters: α and λ . The default values in the PyTorch library [54] are $\alpha = 1.673$ and $\lambda = 1.0507$.

As a result of the tests, the quality of the classification of 86.8% was obtained. In addition, unlike in simulations with the ReLU function, it was possible to indicate which feature map was disabled by the glial network for each class.

It is also possible to optimize the network by removing filters that generate feature maps for which the value of the sigmoid outputs for each class was lower than the threshold assumed for the experiment. The network in this case was optimized by removing filters, where the maximum average value of the output signal of the glial structure (output of sigmoid functions) for a given class was lower than 0.3. However, this procedure required additional training after the pruning process. Removing filters without additional training of the network resulted in a significant reduction in the final classification result.

Table 4.2 shows example values of 20 outputs of the glial driver for the first convolutional layer in the convolution network. From the table we can deduce what filters are required when transferring knowledge between convolutional networks.

All filters that create a specific feature map where the glial driver indicated a value higher than 0.3 can be successfully transferred to a new convolutional network. Example: For class 1 all feature map filters except 7, 8, 19 are required.

4.5 Glial Cells in the Improvement of Network Security

This section presents a technique that uses glial networks to build profiles of computer network users. For these issues it is extremely important to use the feature of this structure that allows us to indicate those parameters (sets of weights) that determine the indication of the final result, e. g., in the case of classification, indication of a particular class at the output. This is due to the fact that in the case of real problems, including the example presented here, the state of information systems frequently changes. In the analyzed example, it should be assumed that the number of users of the computer network is constantly changing, so the number of created profiles and their diversity will also change.

The proposed solution based on the use of glial cells constitutes an innovative method of modifying the structure of neural networks in the context of user identification. It is based on the assumption that it is possible to change the structure of the network without the need for repeated long-term training. This approach has many advantages in the context of user identification. One of them is the ability to modify the network structure depending on the classification of the target group of users. As a result, when a user is removed from the system, it is not necessary to conduct the training process again. The study used a modified convolutional network structure that had previously been used to classify users based on logs of visited websites. This modified network structure, together with glial cells, constitutes the basis for the presented solution. It should be noted that in the case of user classification, a network with 1D convolutional layers is used. The problem is to determine how the glial layer affects the convolutional layer. The networks used to classify text are based on 1D convolutions, so feature maps are vectors rather than two-dimensional matrices as in the case of 2D convolutions. Turning off the channel by the glial cells in this case means turning off one input line. The effect of glial cells on the network is similar to the application of a 2D convolution, since both approaches are aimed at optimising the network structure. The difference between them is mainly due to the implementation of the network and the optimisation process. When glial cells are used, the optimisation process involves removing feature maps and filters from the network, as it was described above. Disabling inactive weights for all users does not adversely affect the functioning of the entire network, as in the case where a 2D convolution is used. It is worth noting that the implementation of glial cells allows for flexible modification of the network structure depending on the needs and groups of users. Removing a feature map or inactive filters for a given task or user does not have negative impact on the operation of the network. Thanks to this approach, the optimisation of the network structure can be implemented in an efficient and dynamic way, allowing the network to adapt to changing conditions and requirements (Fig. 4.9).

Table 4.2 Values of the sigmoid layer for the glial driver for each class

Class	FM1	FM2	FM3	FM4	FM5	FM6	FM7	FM8	FM9	FM10	FM11	FM12	FM13	FM14	FM15	FM16	FM17	FM18	FM19	FM20
1	0.999	0.376	1	0.529	0.823	1	0.096	0.286	0.99	0.371	0.994	0.993	0.998	0.965	0.985	0.497	0.356	0.348	0.105	0.999
2	0.13	0.627	1	0.09	0.601	0.992	0.321	0.995	0.997	0.999	0.979	0.999	0.5	0.323	0.328	0.093	0.999	0.771	0.99	0.987
3	1	0.521	0.819	1	0.101	0.39	0.988	0.373	0.993	0.992	0.997	0.961	0.982	0.497	0.359	0.351	0.11	0.998	0.469	0.984
4	1	0.142	0.567	1	0.089	0.624	0.992	0.315	0.995	0.996	0.998	0.978	0.996	0.5	0.318	0.325	0.091	0.999	0.777	0.989
5	0.985	0.993	0.949	0.976	0.499	0.362	0.358	0.129	0.994	0.557	0.975	0.974	0.989	0.992	0.489	0.992	0.981	0.733	0.989	0.117
6	0.992	0.997	0.961	0.982	0.497	0.359	0.351	0.11	0.998	0.469	0.984	0.984	0.996	0.996	0.489	0.996	0.989	0.685	0.994	0.1
7	0.876	1	0.999	0.997	0.083	1	0.377	1	0.295	0.76	1	0.08	0.494	0.994	0.333	0.997	0.997	0.999	0.978	0.994
8	0.887	0.999	0.989	0.983	0.127	0.995	0.394	0.999	0.416	0.743	0.999	0.121	0.461	0.981	0.37	0.987	0.985	0.993	0.949	0.976
9	0.963	1	0.994	0.99	0.111	0.998	0.379	1	0.521	0.819	1	0.101	0.39	0.988	0.373	0.993	0.992	0.997	0.961	0.982
10	0.365	1	0.596	0.895	1	0.083	0.36	0.993	0.358	0.996	0.995	0.999	0.971	0.988	0.496	0.343	0.332	0.996	0.337	0.978

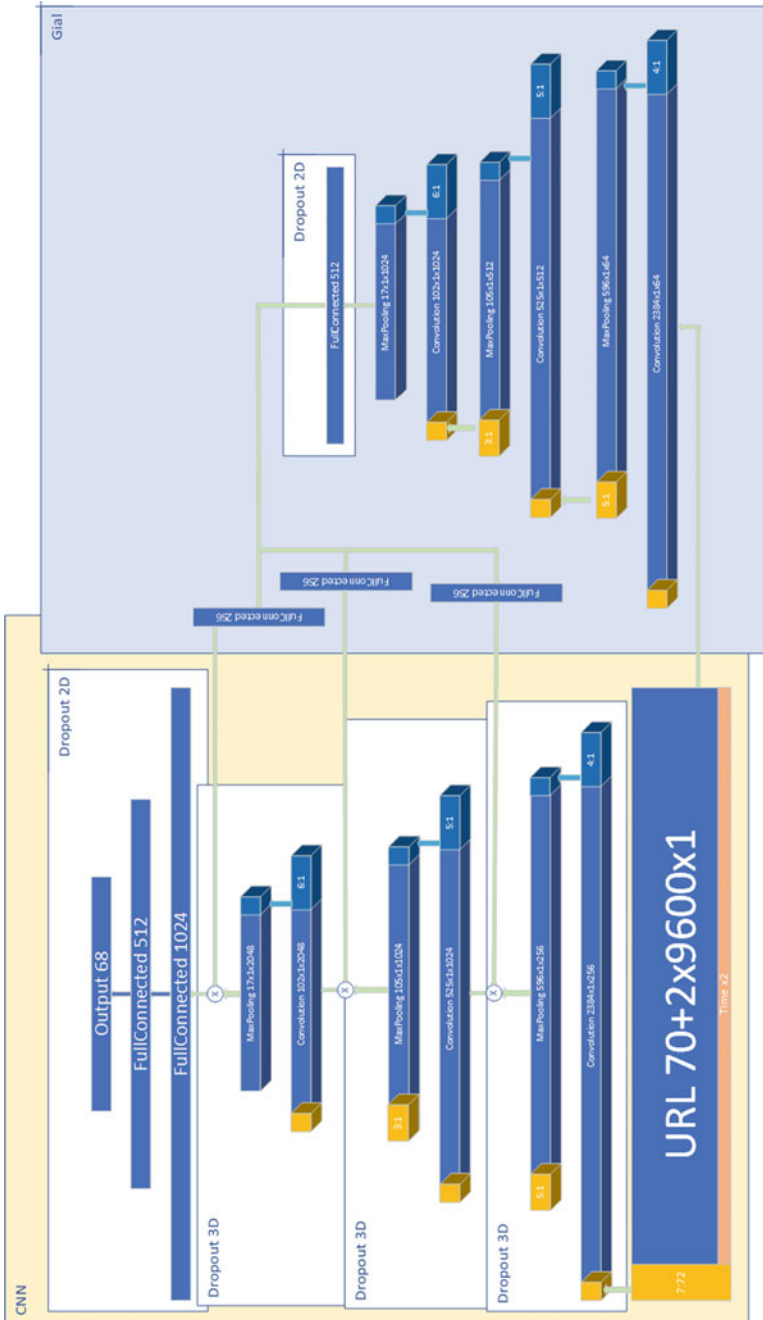


Fig. 4.9 Glial network for users classification based on URLs

The above scheme can be described by Formula (2.32), which is a modification of Formula (2.27). In the equation G_c has been added, which denotes the value of the glial function for each channel. It is always assumed to be a vector with a c dimension and an individual value for each channel.

$$(I * F)_{ij} = \sum_{m=0}^{f_1} \sum_{n=0}^{f_2} \sum_{c=1}^C (I_{i+m, j+n, c} F_{m, n, c} + b) * G_c \tag{4.2}$$

The use of a glial network enables collecting statistics on the operation of the network. Table 4.3 contains information on the use of feature maps for individual users. The analysis of these statistics allows us to assess the effectiveness and efficiency of the applied structure of the glial network and to make decisions regarding the optimization and further training of the network.

In Fig. 4.10 the comparison of training process of the standard Convolutional Neural Networks (CNN) and the Glial CNN is presented. It is worth to see that Glial CNN needs more steps to achieve better results. This is due to the need to train the Glial driver as well.

Table 4.3 The possibility to optimize the glial network for classification of users based on URLs (the output values of the glial driver are below the threshold of 0.3)

Layer No.	Number of feature maps	Number of feature maps that can be removed	Number of active feature maps per user
1	256	6	28
2	1024	41	33
3	2048	67	12

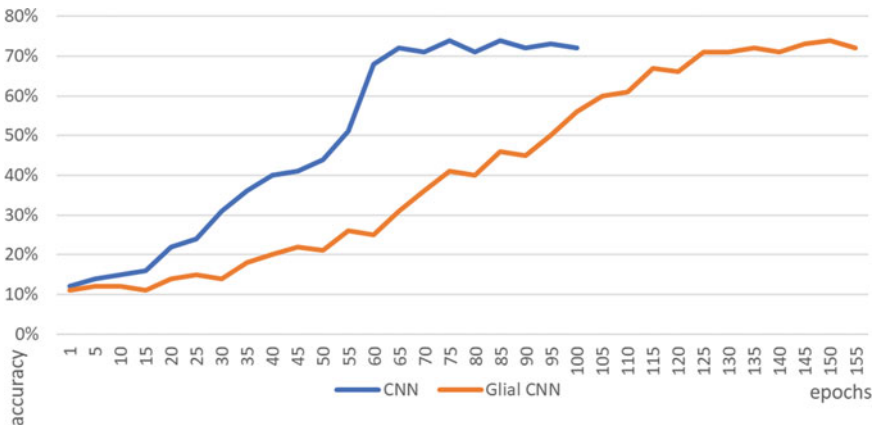


Fig. 4.10 Comparison of the learning process of the standard convolutional neural network (CNN) and the Glial CNN on the example of profiling network users

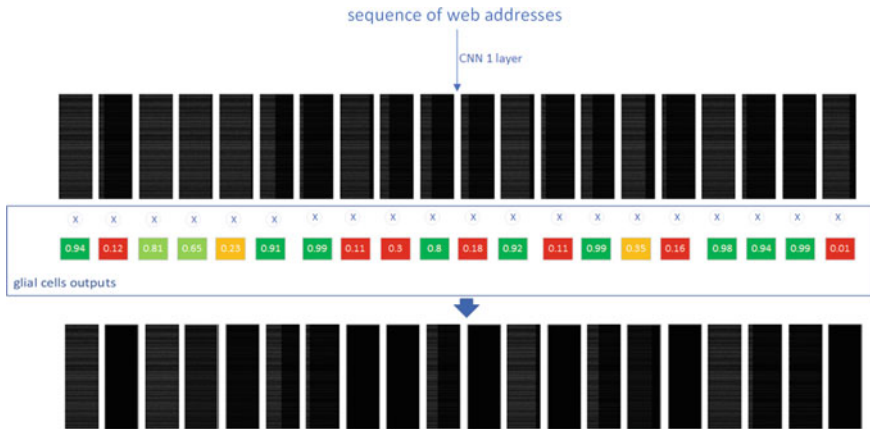


Fig. 4.11 Visualization of the outputs of the first layer of the glial network for URL classification

Based on the data presented in Table 4.3, it can be concluded that the entire network can be reduced by 73 feature maps, which is reflected in its efficiency, An additional advantage is the improved network accuracy, as the network error was reduced by 1.4% percent compared to the original CNN model.

Figure 4.11 shows the effect of glial cells on neuronal baseline. It is easy to see that very small values of these cells practically remove a given cell from the network.

Table 4.4 shows an example result of the sigmoid output values of the glial network for 16 feature maps in the first convolutional layer for the first 10 users. This example shows two feature maps (no. 4 and 8), which can be deleted without affecting the operation of the network. It is easy to see that maps no. 1 and 10 must remain in the network structure to secure its correct operation for each user.

Table 4.5 shows the possibility to select learnt filters for feature maps in order to create a smaller network for selected network users. Conducted tests indicate that all maps for which the values are below 0.3 have a weak impact on the final result of the classification, so this value is critical for removing feature maps from the network. If values with a larger impact are excluded, additional short training is necessary. It is necessary to create a table for each convolutional layer of the basic network. The glial network presented in this section was designed to simplify the trained convolutional network by removing unnecessary neurons. As a practical example of the use of such structures, the concept was tested on a real problem- creating profiles that identify Internet users. With the help of glial cells, filters specific to a particular user of the computer network were indicated. Based on this information, new networks were built to classify users using only those filters that the glial network indicated as relevant. It should be noted that these new structures no longer had glial cells (Table 4.6).

The use of glial cells allows us to optimize the network by removing filters that create irrelevant maps of features in specific sentences. Based on presented experiments, it can be observed that the proposed structures performed very well in

Table 4.4 Comparison of the operation quality of different machine learning methods

Method	Classification quality obtained for data after a one-month break (%)	Precision	Recall	F1	Training time (for the primary string)
Random forest	68.2	0.67	0.66	0.62	72 [s]
CNN network	72.4	0.71	0.68	0.69	241 [min]
LSTM	70.1	0.69	0.61	0.69	481 [min]
BiL LSTM	73.1	0.71	0.72	0.70	587 [min]
GRU	71.4	0.70	0.67	0.68	532 [min]
BiGRU	72.6	0.76	0.71	0.72	549 [min]
CNN OC network	60.2	0.62	0.51	0.51	219 [min]
OC 1 × autoencoder network	68.4	0.63	0.61	0.67	549 [min]
OC 68 × autoencoder network	79.2	0.76	0.77	0.74	1298 [min]
Glial CNN	73.8	0.72	0.69	0.70	348 [min]

the process of classifying images and network users. The experiment presented in Sect. 4.2 contains a method of selecting the number of feature maps in each convolutional layer. This process allows for selecting an optimal structure for solving classification problems using a CNN network.

4.6 Detection of Phishing Threats

Phishing attacks are common and dangerous form of cyberattacks. These are methods in which scammers simulate trusted institutions or organizations in order to extort confidential information from users. A typical example is sending fake emails or text messages containing links to fake websites. These websites are designed to resemble the original ones and mislead users, so that they provide their confidential data, such as passwords, bank account numbers or personal data. Phishing attacks rely on psychological manipulation and exploit users' trust in recognized institutions or individuals. Fraudsters can use a variety of techniques, such as faking logos, e-mail addresses, or the language used in communication, to make their message appear authentic and credible. The effectiveness of phishing attacks depends largely on social engineering and the ability of scammers to manipulate people. Therefore, it is important that users are aware of the risks associated with phishing and exercise caution when using the Internet and sharing their data. Creating fake websites is relatively easier than hacking into a security system. Moreover, phishing campaigns

Table 4.5 Significance values of individual FMs for network operation—the value of the output of the glial driver

User ID	FM 1	FM 2	FM 3	FM 4	FM 5	FM 6	FM 7	FM 8	FM 8	FM 9	FM 10	FM 11	FM 12	FM 13	FM 14	FM 15	FM 16
1	0.50	0.32	0.99	0.09	1.00	0.78	0.99	0.99	0.99	0.99	1.00	0.49	1.00	0.99	0.91	0.06	0.07
2	0.50	0.33	0.33	0.09	1.00	0.63	0.49	0.99	1.00	1.00	1.00	0.49	1.00	0.99	0.83	1.00	0.07
3	0.50	1.00	0.36	0.13	0.99	0.56	0.97	0.08	0.99	0.99	0.99	0.49	0.99	0.98	0.99	0.99	0.12
4	0.50	0.36	1.00	0.11	1.00	0.47	0.98	0.98	0.08	1.00	1.00	0.49	1.00	0.99	0.69	0.99	0.10
5	0.50	0.98	0.36	0.13	0.99	0.46	0.49	0.98	0.99	0.08	0.08	0.49	0.99	0.98	0.64	0.08	0.12
6	0.50	1.00	0.99	0.11	1.00	0.97	0.99	0.99	1.00	1.00	1.00	0.49	1.00	0.99	0.69	1.00	0.09
7	0.50	0.37	0.64	0.12	1.00	0.34	0.98	0.98	0.99	0.99	0.99	0.49	0.99	0.99	0.99	0.99	0.12
8	0.50	0.34	1.00	0.09	1.00	0.43	0.99	0.99	0.49	1.00	1.00	0.49	1.00	0.99	0.67	1.00	0.08
9	0.50	0.64	0.49	0.09	1.00	0.77	0.99	0.99	0.08	1.00	1.00	0.49	1.00	0.99	0.92	1.00	0.08
10	0.50	0.32	0.31	0.07	1.00	0.57	0.99	0.99	1.00	1.00	1.00	0.49	1.00	1.00	0.80	1.00	0.06

Table 4.6 Number of active feature maps for individual users

User ID	Number of filters in the first glial driver layer necessary for the operation of the neural network	Number of filters in the second glial driver layer necessary for the operation of the neural network	Number of filters in the third glial driver layer necessary for the operation of the neural network
1	163	119	62
2	143	131	91
3	145	140	59
4	136	121	111
5	156	148	98
6	198	124	76
7	148	120	62
8	145	98	86
9	121	111	63
10	186	120	111
11	133	131	84
12	164	142	125
13	123	101	92
14	127	122	101
15	196	151	89
16	97	88	74
17	125	102	91
18	210	189	94
19	124	91	102
20	197	127	125
21	167	144	103
22	147	131	122
23	136	98	78
24	122	101	99
25	187	175	117
26	124	112	97
27	125	98	85
28	111	99	85
29	115	91	57
30	119	93	75
31	126	119	85
32	184	176	103
33	126	101	77
34	155	91	84
35	126	121	122

(continued)

Table 4.6 (continued)

User ID	Number of filters in the first glial driver layer necessary for the operation of the neural network	Number of filters in the second glial driver layer necessary for the operation of the neural network	Number of filters in the third glial driver layer necessary for the operation of the neural network
36	245	165	123
37	126	101	91
38	213	122	99
39	124	97	82
40	125	95	89
41	184	117	100
42	184	141	96
43	211	188	177
44	206	142	129
45	184	174	142
46	104	99	101
47	186	123	92
48	164	122	89
49	193	188	106
50	154	122	92
51	196	151	121
52	185	181	99
53	136	122	97
54	111	98	78
55	197	192	127
56	137	92	97
57	196	155	126
58	185	124	97
59	215	166	133
60	218	129	151
61	187	92	88
62	108	91	70
63	127	89	79
64	173	122	89
65	129	82	92
66	219	142	127
67	216	198	188
68	188	176	155

can be conducted relatively cheaply from anywhere in the world, taking advantage of the openness and anonymity of the Internet. Fake websites always contain logotypes and texts that are very similar to the original ones, which is intended to mislead users. According to the Phishing Activity Trends Report 2021 prepared by the Anti-Phishing Working Group (APWG), the number of reported cases of phishing attacks in 2020 reached a record high, exceeding 241,000. According to the report “The State of Phishing 2021” of the organisation Cofense, in 2020 an average of 57% of surveyed companies experienced at least one phishing attack, and 12% of companies were attacked every day. Also interesting is the “2021 Verizon Data Breach Investigations Report”, which states that phishing was a major trigger of data breach incidents in 2020, and attacks of this type accounted for 36% of all data breach incidents. According to the “Phishing and Fraud Report” published by RSA Security, in the first quarter of 2021 there was a 23% increase in the number of phishing attacks compared to the previous quarter. These statistics show that such attacks are still widespread and pose a significant threat to users and businesses. Criminals are constantly improving their methods. Therefore it is important to be aware and careful when dealing with unfamiliar e-mails and suspicious-looking websites. It is worth noting that most ransomware [55] is delivered by phishing methods. Very often even experienced users are not able to check the URLs they visit carefully. These types of attacks mainly affect websites that present significant value to criminals. The main target of attacks are websites which process data for bank transfers. Therefore, financial institutions are particularly interested in blocking phishing attempts. Many users believe that the HTTPS protocol guarantees safety. One of the recent Phishlabs reports [56] shows that in the third quarter of 2018 as many as 49% of phishing websites used SSL certificates. The most common method of attack is the registration of domains with a slightly changed structure (typosquatting) [57]. It includes omitting or adding a single character in the address, registering a domain without a dot after www (wwwdomainname) or adding a hyphen (www-domainname, domainname-anything). Phishing attacks include bitsquatting [58], which consists in changing one bit in the address string. When the address is clicked by the user, it is copied several times, among others:

- By the TCP/IP stack from kernel to user module,
- By the browser while HTML parsing,
- While creating a representation of the DOM tree,
- When creating an http request,
- By the API during DNS resolution.

The attacker expects that an error of writing a string of bits in memory will be encountered at any point while copying. If the error causes one bit to be reversed, the user is directed to an unwanted website. In this case, the only way to avoid the attack is an additional system that checks whether the string definitely belongs to the expected website. Another method is a homographic attack [59]. It transforms characters into ones with a similar appearance, for example, large “i” and small “l” (l), zero and O, etc. The user does not notice the difference in the link containing this type of conversion. Persons who have a visual impairment are particularly vulnerable

to this type of attack. In this technique, attackers can also change one character to two ('m' to 'rn', 'ci' to 'G', 'cl' to 'd' and the other way round). Another method is the representation described in [60]. It consists in representing Unicode characters with a set of characters allowed in the name, i.e. letters, numbers, and ASCII hyphens.

Anti-phishing systems use a variety of methods to detect and prevent attacks. We can mention solutions based on content analysis, checking whether the address is included in the "black list", heuristics and others. Content analysis in anti-phishing systems involves checking e-mails, websites, and other communications to identify suspicious elements, such as URLs, improperly formatted messages, or false logotypes. The most common methods using "black lists" make use of databases containing known and potentially harmful URLs, domains, e-mail addresses and other identifiers. They compare received messages and visited websites with the black list to detect potential threats. However, websites that extort information are active for a very short time, so a black list is usually useless. Systems based on a black list are not able to detect new and so-called zero-day attacks and therefore require frequent updates. As already mentioned, the number of phishing attacks is constantly growing, so it is also not possible to build a permanent (unchanging) black list. It should also be borne in mind that comparing URL strings requires significant computing resources of the system. Therefore machine learning methods are better at this task than a simple black list. There are also anti-phishing systems that use heuristic algorithms to analyze and evaluate suspicious activity. They may include checking for suspicious patterns in the content of messages, identifying unknown or modified elements, and assessing risk based on a variety of factors. An important element in improving security is user education and training on how to recognize and avoid phishing attacks. Educated users are able to display warnings and tips directly, which helps them make informed decisions and avoid traps. These methods are often implemented together in complex anti-phishing systems to provide the highest level of protection against phishing attacks. A huge advantage of anti-phishing methods using AI algorithms is the range of data on which the developed systems must operate. This book focuses only on the analysis of the URL content, omitting the content of the entire website. The tests were conducted using phishing addresses downloaded from the PhishTank portal and addresses not used in attacks, made available by the Crawl Foundation portals [61], the Moz Top 500 [62] and Alexa [63]. On the day of the creation of the training and testing string, the PhishTank database contained approximately 11,000 addresses. This set was completed with the same number of randomly selected addresses from other databases. To save as much space for encoding characters as possible, each URL was stripped of the protocol symbol: `http://` or `https://`. Table 4.7 shows examples of portal addresses used in experiments.

The identification of phishing addresses was performed using a convolutional network, which is a popular tool in the field of analysis and classification of text data. The process of converting input data was based on previous research aimed at analyzing text data in the context of identifying phishing addresses. Within this study, phishing addresses were encoded using 70 different characters. However, unlike in previous experiments, the input was encoded with 512 characters. At the time of the study, only 14 addresses were longer. In practical applications of convolutional

Table 4.7 Examples of phishing addresses

1	http://www.yourbank-login.com
2	http://www.paypal-support.net
3	http://www.apple-verification-login.com
4	http://www.netflix-update-account.xyz
5	http://www.microsoft-security-alerts.info
6	http://www.amazon-rewards-claim.com
7	http://www.icloud-login-security.net
8	http://www.google-account-update-login.org

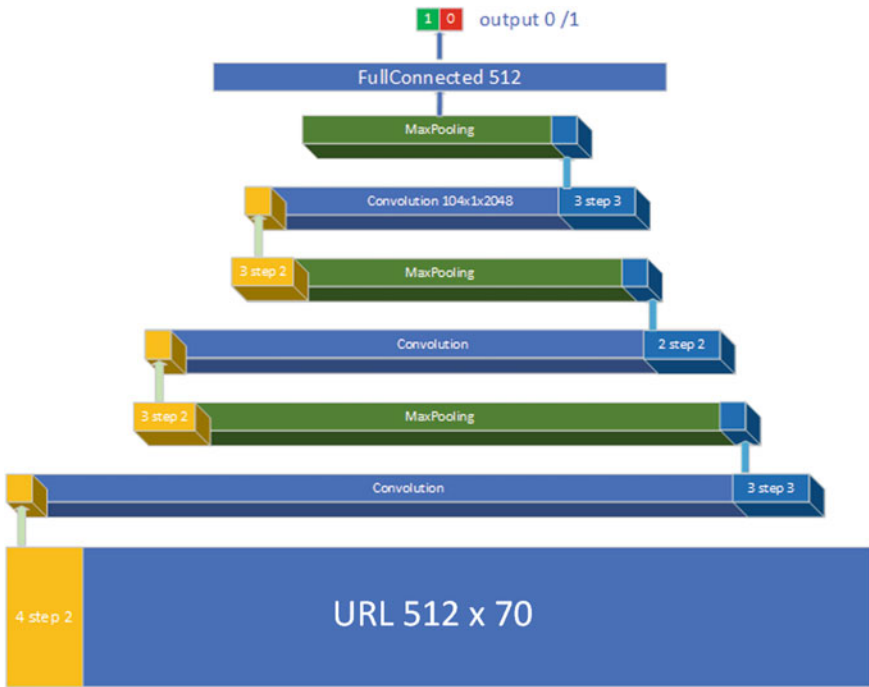


Fig. 4.12 Diagram of the network used to identify phishing websites

networks, it may happen that the address is longer than the available network input space, which should be taken into account when analyzing and classifying data. The best results in identifying phishing addresses are achieved using a specific network structure, which is shown in Fig. 4.11. This network structure was optimal in terms of performance and efficiency in the classification of phishing addresses, taking into account various aspects of textual data analysis (Fig. 4.12).

Table 4.8 The structure of CNN used to solve phishing problem

Layer No.	Layer description
1	Input with a size of: 70×256
2	1D convolution with ReLU activation, 24 feature maps, 1D filter, width 4, step 2
4	Maxpooling 1D filter, width 3, step 3
5	1D convolution with ReLU activation, 64 feature maps, 1D filter, width 3, step 2
6	Maxpooling 1D filter, width 2, step 2
7	1D convolution with ReLU activation, 128 feature maps, 1D filter, width 3, step 3
8	Maxpooling 1D filter, width 3, step 3
9	Dense layer 512 with ReLU activation
10	Output 1 of 2

A diagram of the applied convolutional network is presented in Table 4.8. The structure consists of three convolutional layers connected with maxpooling layers. The last two layers are dense layers which perform the final classification.

The above structure allowed for a correct classification of 98.8% of addresses. The above structure was modified by adding a glial structure. The idea of this study was to find the optimal dimensions of convolutional layers. The network diagram is shown in Fig. 4.13.

The convolutional layers were extended by 128 feature maps each. Table 4.9 shows how the dimensions changed during the training of the network with a glial driver. The optimal dimension of the structure was achieved after 5 iterations. The training procedure was conducted in a manner similar to the one in the previous section. During the 6th and 7th iteration a significant decrease in the correctness of the classification performed by the network could be observed.

Table 4.10 presents a comparison of its operation with the operation of other structures which are most commonly described in the literature, which make use of the PhishTank database.

The presented solution using convolutional networks reached the highest effectiveness. Obviously, a network should be additionally trained and updated based on new addresses used in attacks, so that the desired effectiveness can be maintained. However, the experiments presented in this section prove almost infallible effectiveness in identifying phishing addresses.

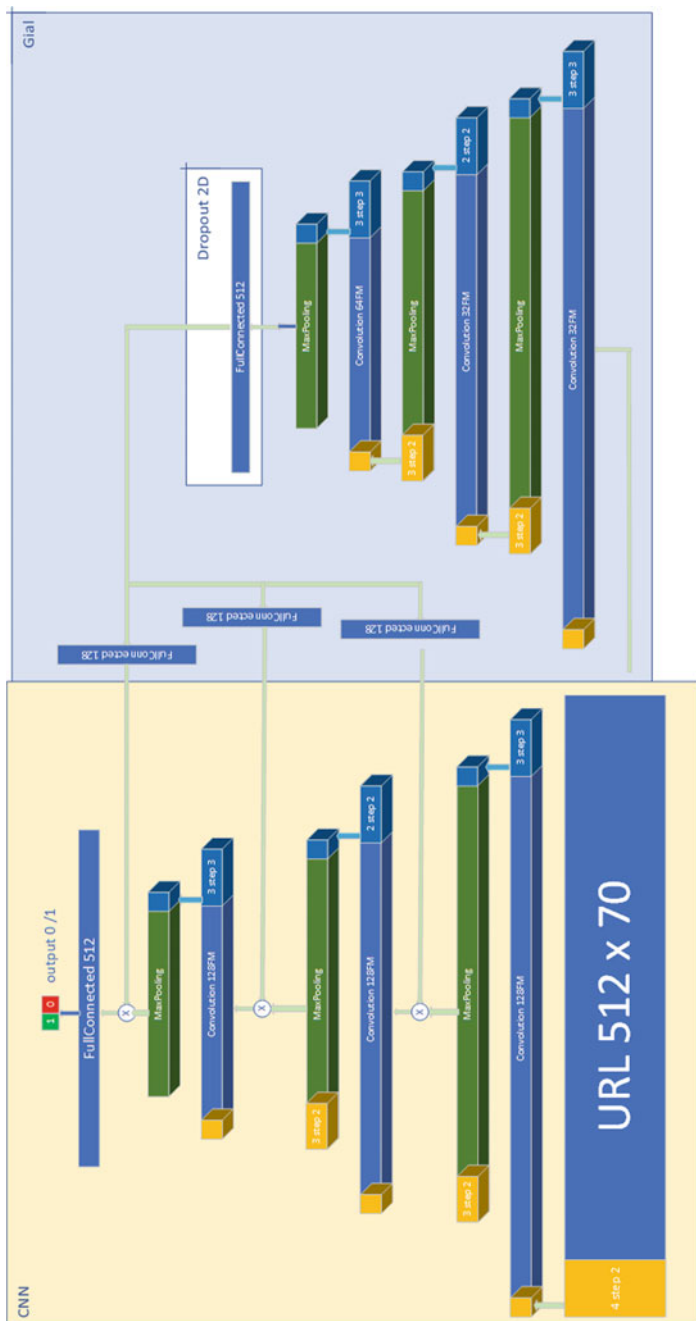


Fig. 4.13 Diagram of a gial network designed to detect phishing addresses

Table 4.9 Course of a network training process

Number of iteration of the training process	Size of layer 1	Size of layer 2	Size of layer 3	Efficiency [%]	Training time glia [s]	Training time CNN [s]	Switched off FM, layer 1	Switched off FM, layer 2	Switched off FM, layer 3
1	128	128	128	95.17	89	92	11	8	9
2	117	120	119	97.43	88	91	27	31	28
3	90	89	91	98.15	73	78	28	16	12
4	62	73	79	99.43	71	67	22	25	18
5	40	48	61	99.86	69	64	0	16	12
6	40	32	49	98.31	70	68	0	7	1
7	40	25	48	86.8	68	63	0	0	1

Table 4.10 Comparison of other solutions used to identify phishing addresses

Method of classification	Correctly classified addresses in the test set (%)
Random Forest [64]	93.40
Recurrent networks LSTM [64]	98.70
SVM [65]	95.80
CANTINA+ [66]	92.00
SVM [67]	99.00
Td-idf [68]	99.62
The convolutional network presented in this book	99.86

Chapter 5

Anomaly Detection Using Sequence Patterns



Analysis of system logs plays an important role in improving the security of IT systems. System logs are records of events and activities that take place in the system, such as logging in, running applications, accessing files, network connections, etc. Analysis of system logs can provide valuable information about user activities and behaviors, detect suspicious or unauthorised activities, and identify potential threats to system security. Analysis of system logs allows system administrators to identify incorrect or unauthorised activities that may indicate attacks or security breaches. By monitoring and analyzing logs, we can quickly detect unwanted activities, such as hacking attempts, suspicious logins, or DDoS attacks [69, 70]. Analysis of system logs allows us to detect irregularities or unusual patterns of behavior. It can include unexpected changes in user activity, unusual file transfers, suspicious network connections, etc. Detecting such anomalies can help identify potential threats or security incidents related to the behavior of the systems. Another important aspect is the reconstruction of events. By carefully checking the event log entries, it is possible to reconstruct the sequence of events when an incident or security breach occurs. By analyzing logs, we can reconstruct the path of the attack, identify sources and methods of the attack, and determine the scale and extent of the incident. It is important for tracking adverse activities and taking appropriate corrective actions. It is also possible to monitor compliance in the context of safety regulations and policies. By analyzing logs, we can verify if systems and users operate in accordance with specific requirements and regulations. This may include monitoring access to sensitive data, adherence to password policies, network security, etc. It is important to configure appropriate tools for collecting, storing and analyzing system logs. Automation of log analysis and the use of tools and technologies such as SIEM (security information and event management) [71] can facilitate the process of monitoring and detecting security incidents.

Security of IT systems is associated not only with protection against deliberate attacks by hostile users, but also against errors in the implementation or installation of systems. Administrators use system event logs to detect problems. Based on them,

they can determine the cycles of events that preceded a failure, or build event profiles that describe a correct operation of the system. The problem here is the analysis of large amounts of data. Often the complexity of the problem is so great that analyzing it becomes impossible even for a proficient administrator. Searching for event cycles or building profiles of properly functioning systems cannot always be done with the help of well-known sequential data mining algorithms, e.g., GSP or Apriori [72, 73], since events which build them rarely occur in the same sequence. Very often, several occur in an alternating way or do not occur at all, which does not mean an error of the system operation. So the goal is to find an “approximate” sequences of events.

System logs are an example of sequential data. Sequential data is data in which items are ordered in a specific sequence or order. It means that each data element has its place and occurs after the previous element, and before the next one. Examples of sequential data can be texts, sounds, time signals, genetic sequences, time logs, financial transactions and others. In the case of sequential data, order is relevant for their analysis and understanding. For example, in text analysis, word order is important for understanding the context and meaning of the text. In sound analysis, the order of sound samples is necessary to reproduce the sound in an appropriate sequence.

Processing of sequential data requires special techniques and models that take into account its sequential nature. Examples of such techniques are sequential models such, for instance Recurrent Neural Networks (RNN), convolutional neural networks (CNN) with sequential layers, Markov modelling and hidden Markov models (HMM). The solution presented in this book is based on the market basket analysis. Market basket analysis, also known as association analysis, is a data analysis technique the purpose of which is to discover the relationships between elements in a set of transactional or sequential data. It is based on identifying frequent sets of elements, that is groups of elements that often occur together in these data. Market basket analysis is designed to understand consumer behavior patterns, product preferences, and other relationships between different elements. The basic element of the market basket analysis is the concept of a “basket”, which represents a collection of items, such as products in a store, items in an online order, or activities in a time sequence. Transactional data is collected in the form of baskets, with each basket containing a set of elements. Market basket analysis focuses on identifying frequently occurring sets of elements, that is, such combinations whose frequency of occurrence is higher than a certain established threshold. For example, in the context of a retail store, a market basket analysis can help identify sets of products that are often bought together. This can lead to the discovery of associative rules, such as “When a customer buys bread, they usually also buy butter”, which can be used for marketing purposes, product recommendations or personalization of offers for customers.

Association analysis is based on various algorithms, such as Apriori, FP-Growth, ECLAT and others, which identify frequent sets of elements and generate associative rules based on this data. Association analysis can provide valuable insights into customer preferences, sales strategies, product improvements, and other business aspects. The insights from the association analysis can be used to make business

decisions, such as product placement in the store, customisation of offers, creating product packages, pricing, and more. Market basket analysis is widely used in areas such as retail, e-commerce, marketing, product recommendations, customer behavior analysis and many others where understanding the connections between the elements is crucial. It is worth noting that there are many other algorithms and techniques available in the field of market basket analysis and discovering of association rules. The choice of an appropriate algorithm depends on the specific case and research requirements. In the following sections, we will discuss each of the algorithms in detail, presenting their mechanism, as well as strengths and weaknesses.

Market basket analysis and discovering of association rules are extremely useful in areas such as marketing, e-commerce, product recommendations, content personalisation, and others. With these techniques we can discover important patterns and dependencies in our data, which enables us to make better decisions and improve our business strategies.

The Apriori algorithms are popular market basket analysis algorithms that are used to identify frequent sets of items in sequential data, such as transactions or shopping carts. The operation of these algorithms is based on the concept of support and confidence.

1. **Support:** Support specifies the frequency at which a set of elements occurs throughout the data set. In the Apriori algorithms a minimum support is established, i.e. a minimum percentage or number of occurrences for a set to be considered frequent. It is defined by (2.45) where support X refers to T as part of the transaction t in the set containing the element X

$$supp(X) = \frac{|\{t \in T; X \subseteq t\}|}{|T|} \quad (5.1)$$

2. **Confidence:** Confidence measures how often another set of items appear with the given set in transactions. It expresses the probability that if a customer buys one set of products, they will also buy another set. Confidence is calculated based on the number of transactions that contain both sets of elements.

The operation of the Apriori algorithm can be described in the following steps:

1. **Data preparation:** Data is represented as a collection of baskets, where each basket contains a set of products.
2. **Generating frequent single-element sets:** The algorithm starts by identifying frequent single-element sets (products that are often found in the baskets). Support is calculated for each product. Then products that do not meet the minimum support criterion are removed.
3. **Generating candidates for larger sets:** Based on the frequent single-element sets, candidates for larger sets are generated, adding one element to existing sets. Then support for product sets is calculated and product sets that do not meet the minimum support criterion are removed.

4. Generating association rules: Based on the frequent sets, associative rules for the relationships between sets of products are generated. The rules are evaluated based on the confidence value and the minimum confidence established before.

The Apriori algorithms allow for identification of frequent sets of products and generation of associative rules that can provide information about customer preferences, product recommendations or marketing strategies.

5.1 Neural Networks in Sequence Pattern Detection

Neural networks are successfully used to detect sequential patterns in data. In particular, Recurrent Neural Networks (RNN) are specially designed for processing sequential data and have the ability to recognize and model the relationships between elements in a sequence. In the case of sequential data such as time sequences, texts, sounds or gestures, neural networks can be used to identify hidden patterns, predict sequential elements, generate new sequences, translate languages, analyze emotions and perform many other tasks related to sequential processing. Recurrent neural networks, such as long short-term memory (LSTM) and gated recurrent unit (GRU), have memory mechanisms that allow them to store information about earlier elements of a sequence and use it to predict subsequent elements. As a result, they are able to model time dependencies and detect sequential patterns in the data. For example, when analyzing sentiment in text, neural networks can learn to recognize word sequences or sentence structures that indicate a positive or negative attitude. In the case of speech recognition, neural networks can detect sound patterns that correspond to individual words or phonemes. In general, neural networks have the ability to detect and model sequential patterns in data, which makes them a powerful tool in sequential data analysis and many other fields, such as natural language processing, speech recognition, sound processing, time sequences prediction, and many more.

However, it should be remembered that the search sequences must be given at the training stage. The classical approach to training neural networks does not enable a recognition of a sequence that was not included in the training string. Due to time and hardware limitations and many potential combinations it is not possible to generate all training sequences. Attempts to train traditional convolutional networks on sequential data, unfortunately, have not yielded positive results.

5.2 Generating Sequences

The generation of sequential patterns can be performed in various ways, depending on the specifics of the data and the needs. In the experiments random generation and supervised training were applied. In the case of standard sequential data exploration, in the process of simulation studies, it is assumed that the effectiveness of the applied

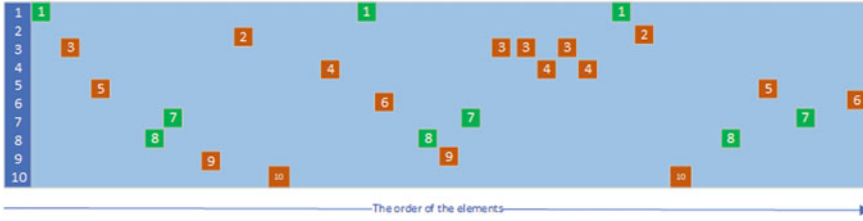


Fig. 5.1 Examples of sequences created by the generator for the purpose of verifying the operation of neural networks in the problem of finding sequence patterns

methods is equal to 100%. Thus, only the complexity of the algorithms and, indirectly, the time of information processing are assessed. In the case of soft computing methods, which include neural networks, we also analyze their effectiveness. The aim of the conducted research was to verify the possibility of using the structures of convolutional networks with optimally chosen structures to detect sequences that were not included in the training data. In other words, the network needs to capture the general idea and find repeatable patterns instead of remembering specific sequences occurring in a dataset. Moreover, any type of sequential data can be converted from a dictionary into numbers. A generator of synthetic random data sequences was developed. Each position was a number from 1 to 1000—the maximum value that we can encode in a given U-Net network. One training example consisted of smaller transactions with a hidden repeated sequence of characters (green colour in Fig. 5.1) and other numbers drawn from among numbers which were not used to create the sequence. Each transaction was followed by a break of additional random characters that were not used to create the sequence. An example sequence is shown in Fig. 5.1.

The parameters applied for generating sequences are:

- size of the input convolutional network (dictionary size),
- maximum space between sequence elements (blue parts in Fig. 5.1),
- maximum and minimum length of the sequence (numbers in green in Fig. 5.1),
- maximum distance between sequences (yellow fields in Fig. 5.1).

Using this type of generator has several key advantages for neural networks. During the training we want to find out what sequence the network should return and calculate the training error. The data was encoded using a one-hot vector. The input signal for the convolutional network (and output network) is a two-dimensional matrix. In algorithms such as GSP or SPADE there is no limit to the number of elements where a sequence is cancelled. In the case of the proposed method, it must be assumed in advance. The proposed method reveals sequential patterns. Standard algorithms, such as BIDE, have a user-defined threshold called minimum support (a value from the range [0,1]). In the proposed method we cannot set this parameter, and the only possibility is to prepare appropriate training data.

5.3 U-Net Network in Sequence Pattern Recognition

In the presented issue, an attempt was made to create a universal solution that enables finding hidden sequence patterns using convolutional networks. The U-Net [37] structure is a special convolutional network architecture used in the previous parts of the monograph. It has a specific neural network structure and is used in the field of image processing and semantic segmentation. It was proposed by Olaf Ronneberger, Philipp Fischer and Thomas Brox in 2015 and has since been used in many tasks related to the analysis of medical images. The U-Net network is characterised by a symmetrical structure of the encoder and decoder, which allows for effective extraction of the image features at different resolution levels, and then accurate reproduction of the image with appropriate details at higher resolution levels. In the U-Net architecture, the encoder consists of a series of convolutional layers that are used to extract image features. The decoder consists of deconvolutional layers, which are used to reproduce the image in the original resolution. Importantly, information from the corresponding encoder levels is added to the decoder levels, which allows for effective communication of context and local details. The U-Net network is often used in image segmentation tasks where the goal is to assign labels (e.g. pixel classes) to different areas of the image. Thanks to its architecture, U-Net can precisely locate and segment objects in medical images, such as organs, tumours, or pathological lesions. The U-Net network has gained popularity in the medical field, but it is also applied in other areas, such as image recognition, optical text analysis, satellite image segmentation, and many other image processing tasks for relatively small training datasets. It was this feature that was the main reason for choosing this structure (Fig. 5.2).

The 2×2 max-pooling operation is used to reduce the map of objects by selecting the elements with the highest value. Un-pooling is the reverse of max-pooling [74]. There are many ways to perform this operation, for example, creating an additional filter in which the layer is calculated by “diluting” the image, or by describing each value from the input map function with zeros. You can also duplicate any value from size 1×1 to size 2×2 . Maps from the encoder are also copied to the decoding part, creating a single 3D matrix on which the convolution and layers are performed simultaneously after increasing the pool.

When designing a U-Net architecture, we need to determine the size of the input and output two-dimensional matrices. With this architecture, the input and output data have the same size m , which corresponds to the number of encoded characters (size of the dictionary) or the number of unique elements that make up the sequence base. There is no maximum number of elements in the analyzed data (a window that searches for sequences). As with black and white images, only one input channel is used. The goal is to get only those numbers that form a repeatable sequence at the U-Net output. The network uses zero filling, so that the size of the network remains unchanged after the performance of convolutions [109].

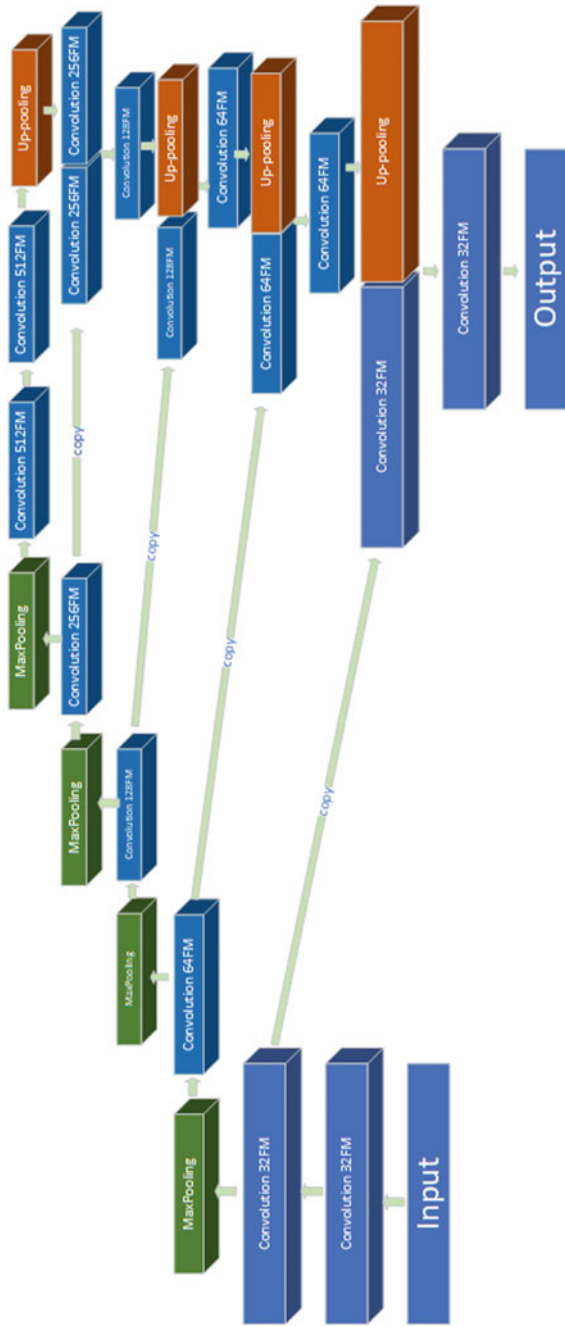


Fig. 5.2 U-Net network structure for detecting hidden sequences

5.4 Network Training for Detecting Hidden Sequences

The U-Net network was trained using the adaptive moment estimation (Adam) algorithm [75]. U-Net is a specific neural network architecture, mainly used in image segmentation tasks. It consists of convolutional layers and pooling layers.

The Adam algorithm is a popular gradient optimisation method that combines the advantages of the momentum and RMSprop algorithms. The main goals of the Adam algorithm are effective adjustment of the training factor and tracking of the history of gradients.

The process of training U-Net with the Adam algorithm has the following stages:

1. Initialisation of weights: U-Net weights are initialised randomly or using specific initialisation strategies.
2. Data preparation: Training data is divided into mini-batches, which allows for more efficient gradient calculations and updating of the network weights.
3. Forward propagation: A mini-batch of data is propagated forward through the U-Net. Predictions are calculated at the output of the network.
4. Calculation of a loss function: The network predictions are compared with real labels and a loss function such as cross entropy is calculated.
5. Calculation of gradients: Gradients of the loss function are calculated using backpropagation. These gradients indicate how the network weights should be adjusted to minimise the loss function.
6. Update of weights: The Adam algorithm calculates the updates of weights, taking into account the momentum of the gradient and the square mean of the gradient. The weights are updated in a direction that minimises the loss function.
7. Repetition of the process: The above steps are repeated for subsequent mini-batches of training data until all mini-batches are used in the training process (one run through the training set is one epoch).

Through iterations of the training process, the Adam algorithm adjusts U-Net weights to minimise the loss function and increase the network's ability in order to accurately segment images. It is worth noting that the Adam algorithm offers adaptive adjustment of the training factor depending on the observed gradients, which contributes to the effective training of neural networks such as U-Net. In the training process, the momentum value was set to 0.9 [76], and the training factor to 1×10^{-3} in the first 2 epochs. It was then reduced to 1×10^{-4} throughout the training process. The choice of the loss function is an extremely important element in the case of U-Net. The Dice coefficient (DSC) [28, 77] defined by (2.46) was used.

$$DSC = \frac{2 \sum_{i=1}^n \sum_{j=1}^m X_{ij} Y_{ij}}{1 + \sum_{i=1}^n \sum_{j=1}^m X_{ij} + \sum_{i=1}^n \sum_{j=1}^m Y_{ij}} \quad (5.2)$$

It is a measure of similarity or overlapping of two sets. It is often used in image segmentation tasks and assessment of quality of segmentation model predictions. In the designed network no normalization or transformation (data augmentation)

of the patterns was applied for training data. Obviously, for the purposes of the analyzed problem it is necessary to give examples of the sequence in its original form, without any modifications. An vital parameter in the training process is the size of the mini-batch. As a result of the simulations, it can be concluded that the best results were achieved using the mini-batch value = 1. In consequence, the network training process became sequential in nature.

Analysis of server system logs using sequences

System logs can be analyzed by searching for sequence patterns, which refers to searching for relevant information from logs generated by a server or computer system. System logs contain records of events and activities related to system operation, such as logging errors, network requests, performance information, monitoring, etc. Analysis of these logs is essential for monitoring, diagnostics, security, and system optimisation. Sequential analysis in the context of system logs is based on investigation of the sequence of events or activities in order to identify patterns, anomalies, errors, or other relevant information. It may involve detecting abnormal behavior, attacks, performance issues, application errors, as well as trend analysis and forecasting. Sequential analysis of system logs can provide valuable information about system performance, help detect problems, improve security, optimize performance, and support decision-making within system management. It is an important tool in areas such as systems administration, IT infrastructure Monitoring, security event management, and data analysis. Various log analysis techniques can be distinguished, such as analysis of time relationships between events to detect anomalies or time patterns. The use of statistical models or machine learning to model sequences of events and predict future events. Analysis of sequences of events related to specific applications or scenarios in order to identify performance issues, bugs, or optimisation opportunities. The presented sequence detection system was verified in a real-life environment—the problem of analyzing entries in the HDFS server log with [78]. In this data set, the task was to detect a sequence of events that preceded the occurrence of the error. An example log entry is shown in Table 5.1.

The study included an analysis of groups of events, each of them containing 32 entries before the information with the label “ERROR”. Log pre-processing with the U-Net network is shown in Table 5.2. The data set contained 563 entries labelled

Table 5.1 Selected event log entries

<i>081110 222826 15661 INFO dfs.DataNode\$DataXceiver: Receiving block blk_6270246515882750563 src: /10.251.127.243:44841 dest: /10.251.127.243:50010</i>
<i>081110 222825 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.107.196:50010 is added to blk_2055760797155771987 size 67108864</i>
<i>081110 222827 16033 INFO dfs.DataNode\$PacketResponder: PacketResponder 2 for block blk_-7146089275359223485 terminating</i>
<i>081,111 023,107 17,687 ERROR dfs.DataNode\$DataXceiver: 10.251.194.147:50010:DataXceiver: java.io.IOException: Block blk_-7877899410184422264 is not valid</i>

“ERROR”. As it was mentioned, all data preceding entries with this label were divided into groups of 32. Grouped log entries were then converted to 512-element input vectors. If a group of 32 events exceeded the size of 512 characters, the events were moved to the next input vector. Events were encoded as a matrix consisting of single one-hot vectors, with a class number specifying the type of event. Thus, the number of events in a single input vector never exceeded 512, which served to maintain the relative size of the U-Net network.

Table 5.2 Sequence of log entries preceding a server error

10.131.0.1 GET	/details.php?id=41	HTTP/1.1,200
10.131.2.1 GET	/js/chart.min.js	HTTP/1.1,200
10.131.2.1, GET	/	HTTP/1.1,200
10.131.0.1 GET	/contestproblem.php?name=RUET%20OJ%20Se r	HTTP/1.1,200
10.130.2.1 POST	/pcompile.php	HTTP/1.1,200
10.131.2.1 GET	/contestssubmit.php?id=42	HTTP/1.1,200
crash error		
10.131.2.1 POST	/pcompile.php	HTTP/1.1,200
10.131.0.1 GET	/details.php?id=41	HTTP/1.1,200
10.131.2.1 POST	/contestsubmission.php	HTTP/1.1,200
10.131.2.1 GET	/details.php?name=Research%20Items&cod=16	HTTP/1.1,200
10.128.2.1 GET	/contestssubmit.php?id=42	HTTP/1.1,200
10.131.2.1 POST	/pcompile.php	HTTP/1.1,200
10.131.2.1 GET	/details.php?name=Research%20Items&cod=16	HTTP/1.1,200
10.128.2.1 GET	/contestssubmit.php?id=42	HTTP/1.1,200
10.131.2.1 GET	/	HTTP/1.1,200
crash error		
10.131.2.1 GET	/details.php?name=Research%20Items&cod=16	HTTP/1.1,200
10.211.2.1.POST	/test.php	
10.131.2.1 GET	/contestssubmit.php?id=42	HTTP/1.1,200
10.128.2.1 GET	/contestssubmit.php?id=41	HTTP/1.1,200
10.130.2.1 POST	/pcompile.php	HTTP/1.1,200
10.128.2.1 GET	/details.php?id=41	HTTP/1.1,200
10.130.2.1 POST	/contestsubmission.php	HTTP/1.1,200
10.130.2.1 GET	/details.php	HTTP/1.1,200
10.131.2.1 GET	/	HTTP/1.1,200
crash error		

Table 5.2 shows the sequence sets in which a WWW server error occurred, together with the events preceding it. Here we can see what set of events led to the error. In this table, the entry marked in orange always occurred before an error was triggered. Thanks to that the administrator was able to debug the server in order find the failure. The presented method of detecting sequences in computer system logs can replace traditional exploration algorithms, such as BIDE. The U-Net architecture demonstrated high effectiveness and speed of operation. A method based on networks with a structure identical to that of U-NET can effectively and quickly analyze large data sets and detect sequences.

Figure 5.3 presents the input data with hidden sequences for the U-NET network. The network's output is shown in Fig. 5.4, where filtered elements that do not fit the sequential pattern can be observed. A significant advantage of the proposed solution is also the fact that the load associated with the calculations can be distributed over many GPUs [79]. The proposed technique is less efficient than standard algorithms (e.g. GSP, SPADE) only for very short sequences. It should be noted that the presented method concerns finding any sequence that does not belong to the patterns from the training data sets (a given sequence may or may not belong to this set). A trained neural network finds new sequences, previously unseen in training data. In the presented version of the algorithm, it is not possible to determine the minimum parameters such as support.

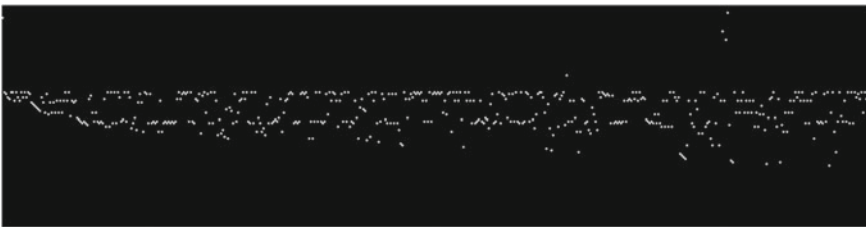


Fig. 5.3 Encoded data from sequences as an image for the U-NET network input



Fig. 5.4 Encoded data from sequences as an image for the U-NET network output

Chapter 6

Concluding Remarks and Challenges for Future Research



This book presents one of the areas of the use of artificial intelligence techniques in tasks related to the security of IT systems. Due to the widespread use of Internet resources and the information collected there (from extremely sensitive data, such as information on human health, to various types of control systems, e.g., drinking water intakes, etc., or databases on the functioning of enterprises, banking systems available on-line), it is an extremely important and very wide field for the application of computational intelligence techniques. One may even be tempted to say that it is practically impossible, due to the amount of data necessary to be processed in an extremely short time (real time), to operate IT security systems without the use of the latest discoveries in the field of AI. This monography focuses mainly on the use of convolutional neural networks and decision trees for tasks related to the creation of computer network user profiles (Chap. 3) and the detection of phishing threats (Sect. 4.6). An extremely important element of this research is the fact that it was carried out using data from real firewall devices based on authentic network traffic from a wide area WAN. The extraordinary effectiveness of the proposed methods should be emphasized. It is also worth noting that the solution to the problem of searching for anomalies was carried out with the use of proprietary models of neural networks—the so-called glia. This is one of the first known attempts in the literature to reproduce the latest information on the structure of the human brain. The monograph presents both a proposal for the adaptation of glial cells to convolutional structures known and commonly used, as well as techniques for their training. On the basis of the conducted experiments, it has been shown that:

1. The use of these cells contributes to the improvement of the quality of convolutional network structures known from the literature
2. The concept of modified convolutional networks, equipped with glial cells, can be used to simplify their structure without sacrificing efficiency, quality of operation

In order to demonstrate the advantages of glial networks, their effectiveness was compared with convolutional networks known in the literature, demonstrating the

advantage of newly developed models. It seems natural to continue research on the possibilities of the proposed glial structures. It should be assumed that the proposed model of artificial networks using additional cells is not the final model. It should also be recognized that the developed technique of training the presented networks does not necessarily close the topic completely. A rather interesting task is also the successful attempt to analyze sequence patterns in the logs of information systems, presented in chapter number 5. Thanks to the use of the proposed technique, it is possible to build a description of the sequence of events that naturally occur in the selected IT system. Please note that the analyzed logs are represented by databases up to 50 GB in size. A characteristic feature of such log sets is that their content is highly variable and dependent on interrelated events in different systems. Therefore, it is not possible to analyze them in real time by a human. Here, too, information from real systems was used. An example of the application of this technique to the analysis of events originating in the HDFS system is presented. It should be emphasized here that the extremely wide possibilities of applying the developed proprietary techniques presented in this monograph in real conditions. They can be adapted to various fields related to the broadly understood security of IT systems: analysis of computer logs, database servers, etc. Of course, the use of the proposed algorithms entails the need to provide high-performance equipment for calculations (training and operation in the recreation mode). Of course, there is a possibility, in the era of virtually unlimited bandwidth of computer networks, to use the capabilities of cloud computing for this purpose. However, it should be remembered that each sending of requests outside of one's own infrastructure carries the risk of leakage of extremely crucial information regarding the configuration of devices, and thus data leakage.

Bibliography

1. R. Leszek, *Metody i techniki sztucznej inteligencji: inteligencja obliczeniowa* (Wydawnictwo Naukowe PWN, 2006)
2. G. Thimm, E. Fiesler, Neural network initialization, in *International Workshop on Artificial Neural Networks* (Springer, 1995), pp. 535–542
3. X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (2010), pp. 249–256.
4. K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on image net classification, in *ICCV* (2015)
5. P. Najgebauer, R. Scherer, L. Rutkowski, Fully convolutional network for removing dct artefacts from images, in *2020 International Joint Conference on Neural Networks (IJCNN)* (2020), pp. 1–8
6. D. Rutkovskaya, M. Pilinsky, L. Rutkovsky, in *Neural Networks, Genetic Algorithms and Fuzzy Systems* (Goryachaya liniya—Telekom, Moscow, 2006), p. 452
7. L. Rutkowski, Generalized regression neural networks in a time-varying environment, in *New Soft Computing Techniques for System Modeling, Pattern Classification and Image Processing* (Springer, 2004), pp. 73–134
8. L. Rutkowski, M. Jaworski, P. Duda, L. Rutkowski, M. Jaworski, P. Duda, Probabilistic neural networks for the streaming data classification, in *Stream Data Mining: Algorithms and Their Probabilistic Properties* (Springer, 2020), pp. 245–277
9. K. Gregor, I. Danihelka, A. Graves, D. Rezende, D. Wierstra, Draw: a recurrent neural network for image generation, in *International Conference on Machine Learning* (2015), pp. 1462–1471
10. J.S. Denker, R.E. Howard, L.D. Jackel, Y. LeCun, Hierarchical constrained automatic learning neural network for character recognition (1991)
11. A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks. *Commun. ACM* **60**(6), 84–90 (2017)
12. Y. LeCun, Y. Bengio, Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks* **3361**(10), 1995 (1995)
13. Y. Kim, Convolutional neural networks for sentence classification, arXiv preprint [arXiv:1408.5882](https://arxiv.org/abs/1408.5882) (2014)
14. S. Abdoli, P. Cardinal, A.L. Koerich, End-to-end environmental sound classification using a 1D convolutional neural network. *Expert Syst. Appl.* **136**, 252–263 (2019)
15. D. Scherer, A. Muller, S. Behnke, Evaluation of pooling operations in convolutional architectures for object recognition, in *Artificial Neural Networks—ICANN 2010: 20th International Conference, Thessaloniki, Greece, September 15–18, 2010, Proceedings, Part III 20* (Springer, 2010), pp. 92–101

16. V. Dumoulin, F. Visin, A guide to convolution arithmetic for deep learning, arXiv preprint [arXiv:1603.07285](https://arxiv.org/abs/1603.07285) (2016)
17. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. *Proc. IEEE* **86**(11), 2278–2324 (1998)
18. D. Yu, H. Wang, P. Chen, Z. Wei, Mixed pooling for convolutional neural networks, in *Rough Sets and Knowledge Technology: 9th International Conference, RSKT 2014, Shanghai, China, October 24–26, 2014, Proceedings 9* (Springer, 2014), pp. 364–375
19. P. Drozda, K. Sopyła, P. Górecki, Different orderings and visual sequence alignment algorithms for image classification, *Artificial Intelligence and Soft Computing: 13th International Conference, ICAISC 2014, Zakopane, Poland, June 1–5, 2014, Proceedings, Part I 13* (Springer, 2014), pp. 693–702
20. J. Cheng, F. Wu, L. Liu, Q. Zhang, L. Rutkowski, D. Tao, in DecGAN: learning to generate complex images from captions via independent object-level decomposition and enhancement, *IEEE Transactions on Multimedia* (2023)
21. P. Czerpak, P. Drozda, K. Sopyła, Semantic web system for automatic plan scheduling, in *New Challenges in Computational Collective Intelligence* (Springer, 2009), pp. 39–50
22. A. Buriro, B. Crispo, M. Conti, Answer Auth: a bimodal behavioral biometric-based user authentication scheme for smartphones. *J. Inf. Secur. Appl.* **44**, 89–103 (2019)
23. P. Ainsworth, in *Offender Profiling Crime Analysis* (Willan, 2013)
24. Z. Xiang, Z. Junbo, Y. LeCun, Character-level convolutional networks for text classification, in *Advances in Neural Information Processing Systems* (2015), pp. 649–657
25. M.F. McTear, Z. Callejas, D. Griol, in *The Conversational Interface* (Springer, 2016)
26. P. Górecki, P. Artiemjew, P. Drozda, K. Sopyła, Categorization of similar objects using bag of visual words and support vector machines. *ICAART* **1**, 231–236 (2012)
27. G. Salton, C. Buckley, Term-weighting approaches in automatic text retrieval, in *Information Processing & Management* (1988), pp. 513–523
28. D. Cournapeau, Scikit-learn Machine Learning in Python, 01 10 2020. [Online]. Available: <http://scikit-learn.org/>
29. M. Amrehn, S. Gaube, M. Unberath, F. Schebesch, T. Horz, M. Strumia, S. Steidl, M. Kowarschik, A. Maier, UI-net: Interactive artificial neural networks for iterative image segmentation based on a user model, arXiv preprint [arXiv:1709.03450](https://arxiv.org/abs/1709.03450) (2017)
30. A.F. Peter, K. Meelis, Precision-recall-gain curves: PR analysis done right, in *NIPS* **15** (2015)
31. J. Bilski, L. Rutkowski, J. Smolag, D. Tao, A novel method for speed training acceleration of recurrent neural networks, Elsevier. *Inf. Sci.* **553**, 266–279 (2021)
32. A. Graves, J. Schmidhuber, Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Netw.* **18**(5–6), 602–610 (2005)
33. Z. Wu, S. King, Investigating gated recurrent networks for speech synthesis, in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2016), pp. 5140–5144
34. P. Liu, X. Qiu, X. Huang, Recurrent neural network for text classification with multi-task learning, arXiv preprint [arXiv:1605.05101](https://arxiv.org/abs/1605.05101) (2016)
35. S. Krzysztof, D. Paweł, Assessing the sentiment of book characteristics using machine learning NLP models, in *International Conference on Artificial Intelligence and Soft Computing* (Springer, 2022), pp. 218–231
36. M. Kuprowski, P. Drozda, Feature selection for airborne LiDAR point cloud classification. *Remote Sens.* **15**(3), 561 (2023)
37. X. Zhang, Y. LeCun, Byte-level recursive convolutional auto-encoder for text, arXiv preprint [arXiv:1802.01817](https://arxiv.org/abs/1802.01817) (2018)
38. O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in *International Conference on Medical Image Computing and Computer-Assisted Intervention* (2015), pp. 234–241
39. W. Shi, J. Caballero, F. Huszar, J. Totz, A.P. Aitken, R. Bishop, D. Rueckert, Z. Wang, Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 1874–1883

40. L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S.A. Siddiqui, A. Binder, E. Muller, M. Kloft, Deep one-class classification, in *International Conference on Machine Learning*, PMLR (2018), pp. 4393–4402
41. A. Talun, P. Drozda, L. Bukowski, R. Scherer, FastText and XGBoost content-based classification for employment web scraping, in *International Conference on Artificial Intelligence and Soft Computing* (Springer, 2020), pp. 435–444
42. A. Conneau, H. Schwenk, L. Barrault, Y. Lecun, Very deep convolutional networks for text classification, arXiv preprint [arXiv:1606.01781](https://arxiv.org/abs/1606.01781) (2016)
43. J. Bilski, L. Rutkowski, J. Smolaż, D. Tao, A novel method for speed training acceleration of recurrent neural networks. *Inf. Sci.* **553**, 266–279 (2021)
44. R. Grycuk, T. Galkowski, R. Scherer, L. Rutkowski, A novel method for solar image retrieval based on the parzen kernel estimate of the function derivative and convolutional autoencoder, in *2022 International Joint Conference on Neural Networks (IJCNN)* (2022), pp. 1–7
45. P. Tabakow, A. Weiser, K. Chmielak, P. Blauciak, J. Bładowska, M. Czyz, Navigated neuroendoscopy combined with intraoperative magnetic resonance cysternography for treatment of arachnoid cysts. *Neurosurg. Rev.* **43**(4), 1151–1161 (2019)
46. A. Czupryn, Terapie komórkowe a prekursorzy neuralne w mózgu dorosłych ssaków. *Kosmos* **65**(2), 163–175 (2016)
47. The CIFAR-10 dataset, University of Toronto, [Online]. Available: <https://www.cs.toronto.edu/~kriz/cifar.html>
48. A. Dosovitskiy, J.T. Springenberg, M. Riedmiller, T. Brox, Discriminative unsupervised feature learning with convolutional neural networks, in *Advances in Neural Information Processing Systems*, vol. 27 (2014), pp. 766–774
49. Microsoft, CNTK Examples: Image/Classification/ConvNet, [Online]. Available: <https://github.com/microsoft/CNTK/tree/release/latest/Examples/Image/Classification/ConvNet/Python>
50. S. Targ, D. Almeida, K. Lyman, Resnet in resnet: Generalizing residual architectures, arXiv preprint [arXiv:1603.08029](https://arxiv.org/abs/1603.08029) (2016)
51. Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le and Y. Wu, Gpipe: Efficient training of giant neural networks using pipeline parallelism, in *Advances in Neural Information Processing Systems* (2019), pp. 103–112
52. A.K. Hinton, V. Nair, Geoffrey, CIFAR-10 (Canadian Institute for Advanced Research), [Online]. Available: <http://www.cs.toronto.edu/~kriz/cifar.html>. [Accessed 2022]
53. Z. Li, D. Hoiem, Learning without forgetting. *IEEE Trans. Pattern Anal. Mach. Intell.* **40**(12), 2935–2947 (2017)
54. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**(1), 1929–1958 (2014)
55. PyTorch, Facebook AI, 01 06 2020. [Online]. Available: <https://pytorch.org/>
56. R. Brewer, Ransomware attacks: detection, prevention and cure. *Netw. Secur.* **9**, 5–9 (2016)
57. E. Volkman, 49 Percent of Phishing Sites Now Use HTTPS (2018). [Online]. Available: <https://info.phishlabs.com/blog/49-percent-of-phishing-sites-now-use-https>
58. T. Moore, B. Edelman, Measuring the perpetrators and funders of typosquatting, in *International Conference on Financial Cryptography and Data Security* (2010), pp. 175–191
59. N. Nikiforakis, S. Van Acker, W. Meert, L. Desmet, F. Piessens, W. Joosen, Bitsquatting: exploiting bit-flips for fun, or profit?, in *Proceedings of the 22nd International Conference on World Wide Web* (2013), pp. 989–998
60. Moz, Moz’s list of the top 500 domains and pages on the web (2019) [Online]. Available: <https://moz.com/top500>
61. A. Costello, Punycode: a bootstring encoding of unicode for internationalized domain names in applications (IDNA), *RFC 3492* (2003)
62. The Common Crawl Foundation, Witryna fundacji Common Crawl, [Online]. Available: <https://commoncrawl.org/>

63. J. Horswell, C. Fowler, *The Practice of Crime Scene Investigation*. (CRC Press, 2004), pp. 45–47
64. Witryna internetowa PhishTank, [Online]. Available: <https://www.phishtank.com/>
65. A.C. Bahnsen, E.C. Bohorquez, S. Villegas, J. Vargas, F.A. Gonzalez, Classifying phishing URLs using recurrent neural networks, in *2017 APWG Symposium on Electronic Crime Research (eCrime)* (2017), pp. 1–8
66. M. Zouina, B. Outtaj, A novel lightweight URL phishing detection system using SVM and similarity index, *Hum. Centric Comput. Inf. Sci.* **7**(1) (2017)
67. G. Xiang, J. Hong, C.P. Rose, L. Cranor, Cantina+ a feature-rich machine learning framework for detecting phishing web sites. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **14**(2), 1–28 (2011)
68. Y. Li, R. Xiao, J. Feng, L. Zhao, A semi-supervised learning approach for detection of phishing webpages. *Optik* **124**(23), 6027–6033 (2013)
69. R. Gowtham, I. Krishnamurthi, K.S.S. Kumar, An efficacious method for detecting phishing webpages through target domain identification. *Decis. Support Syst.* **61**, 12–22 (2014)
70. B. Narmeen, A.S. Jawwad, S. Khaled, DDoS attack detection and mitigation using SDN: methods, practices, and solutions, Springer. *Arab. J. Sci. Eng.* **42**, 425–441 (2017)
71. S.R. Zeebaree, K. Jacksi, R.R. Zebari, Impact analysis of SYN flood DDoS attack on HAProxy and NLB cluster-based web servers. *Indones. J. Electr. Eng. Comput. Sc* **19**(1), 510–517 (2020)
72. A. Perez-Sanchez, R. Palacios, Evaluation of local security event management system vs. standard antivirus software. *Appl. Sci.* **12**(3), 1076 (2022)
73. M. Zhang, B. Kao, Y. Chi Lap, C. David, A GSP-based efficient algorithm for mining frequent sequences, in *Proceedings of IC-AI* (2001), pp. 497–503
74. R. Agarwal, R. Srikant, Fast Algorithms for Mining Association Rules in Large Databases, in *Proceedings of the 20th VLDB Conference* (1994), pp. 487–499
75. L. Xu, J.S. Ren, C. Liu, J. Jia, Deep convolutional neural network for image deconvolution, in *Advances in Neural Information Processing Systems* (2014), pp. 1790–1798
76. D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint [arXiv:1412.6980](https://arxiv.org/abs/1412.6980) (2014)
77. I. Sutskever, J. Martens, G. Dahl, G. Hinton, On the importance of initialization and momentum in deep learning, in *International Conference on Machine Learning* (2013), pp. 1139–1147
78. Z. Zhu, Y. Xia, W. Shen, E. Fishman, A. Yuille, A 3D coarse-to-fine framework for volumetric medical image segmentation, in *2018 International Conference on 3D Vision (3DV)*, (2018), pp. 682–690
79. M. Du, F. Li, G. Zheng, V. Srikumar, Deeplog: anomaly detection and diagnosis from system logs through deep learning, in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), pp. 1285–1298
80. D. Paweł, K. Sopyła, GPU accelerated SVM with sparse sliced EHR-T matrix format. *Int. J. Artif. Intell. Tools World Sci.* **24**(01), 1450012 (2015)
81. M. Minsky, S.A. Papert, *Perceptrons* (MIT Press, 1969)
82. X. Zhang, C. Li, W. Zheng, Intrusion prevention system design, in *The Fourth International Conference on Computer and Information Technology, 2004. CIT'04* (2004), pp. 386–390
83. Security Operating Platform Overview, Palo Alto Networks, 01 10 2020. [Online]. Available: <https://www.paloaltonetworks.com/resources/guides/security-operating-platform-overview>
84. FortiEDR, Fortinet, 01 10 2020. [Online]. Available: <https://www.fortinet.com/products/endpoint-security/fortiedr>
85. BIG-IP Application Security Manage, F5, 01 10 2020. [Online]. Available: <https://www.f5.com/pdf/products/big-ip-application-security-manager-overview.pdf>
86. I. Tripwire, Open Source Tripwire [Online]. Available: <https://github.com/Tripwire/tripwire-open-source>
87. R. Lehti, P. Virolainen, AIDE, [Online]. Available: <https://aide.github.io/>
88. E. Gerbier, AFICK, [Online]. Available: <http://afick.sourceforge.net/>
89. Y. Sun, Y. Chen, X. Wang, X. Tang, Deep learning face representation by joint identification-verification, in *Advances in Neural Information Processing Systems* (2014), pp. 1988–1996

90. P. Zhang, X. You, W. Ou, C.P. Chen, Y.-M. Cheung, Sparse discriminative multi-manifold embedding for one-sample face identification. *Pattern Recogn.* **52**, 249–259 (2016)
91. A. Boles, P. Rad, Voice biometrics: Deep learning-based voiceprint authentication system, in *2017 12th System of Systems Engineering Conference (SoSE)* (2017), pp. 1–6
92. N. Clarke, F. Li, S. Furnell, A novel privacy preserving user identification approach for network traffic. *Comput. Secur.* **70**, 335–350 (2017)
93. M. Gratian, D. Bhansali, M. Cukier, J. Dykstra, Identifying infected users via network traffic. *Comput. Secur.* **80**, 306–316 (2019)
94. J. Hartigan, M. Wong, A K-means clustering algorithm. *Appl. Stat.* **51**, 100–108 (1979)
95. J.A. Hanley, Characteristic (ROC) curvel. *Radiology* **743**(2), 29–36 (1982)
96. M. Lotfollahi, M.J. Siavoshani, R.S.H. Zade, M. Saberian, Deep packet: a novel approach for encrypted traffic classification using deep learning. *Soft. Comput.* **24**(3), 1999–2012 (2020)
97. VPN-nonVPN dataset (ISCXVPN2016), University of New Brunswick, [Online]. Available: <https://www.unb.ca/cic/datasets/vpn.html>
98. J. Sherry, C. Lan, R.A. Popa, S. Ratnasamy, Blindbox: Deep packet inspection over encrypted traffic, in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), pp. 213–226
99. J. Kim, J. Kim, H.L.T. Thu, H. Kim, Long short term memory recurrent neural network classifier for intrusion detection, in *2016 International Conference on Platform Technology and Service (PlatCon)* (2016), pp. 1–5
100. K. C. (1999), <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>. [Online]. Available: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>
101. S. Lawrence, C.L. Giles, Overfitting and neural networks: conjugate gradient and backpropagation, in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* (2000), pp. 114–119
102. S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, *Advances in Neural Information Processing Systems* (2015), pp. 1135–1143
103. M. Babaeizadeh, P. Smaragdīs, R.H. Campbell, Noiseout: A simple way to prune neural networks, arXiv preprint [arXiv:1611.06211](https://arxiv.org/abs/1611.06211) (2016)
104. S.A. Ludwig, Applying a neural network ensemble to intrusion detection. *J. Artif. Intell. Soft Comput. Res.* 177–188 (2019)
105. A. Sarikaya, E. Zraggen, R. DeLine, S. Drucker, D. Fisher, Sequence pre-processing: focusing analysis of log event data, in *IEEE VIS The Event Event: Temporal & Sequential Event Analysis Workshop* (2016)
106. P. Werbos, P. John, in *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences* (1974)
107. W. Youyou, M. Kosinski, D. Stillwell, Computer-based personality judgments are more accurate than those made by humans. *Proc. Natl. Acad. Sci.* **112**(4), 1036–1040 (2015)
108. T.F. Lunt, R. Jagannathan, R. Lee, A. Whitehurst, S. Listgarten, Knowledge based intrusion detection, in *Proceedings of the Annual AI Systems in Government Conference, Washington, DC* (1989)
109. M. Kosinski, D. Stillwell, T. Graepel, Private traits and attributes are predictable from digital records of human behavior. *Proc. Natl. Acad. Sci.* **110**(15), 5802–5805 (2013)
110. Fields and D.R, *Drugi mózg, Rewolucja w nauce i medycynie* (2012)
111. J.S. Denker, R.E. Howard, L.D. Jackel, Y. LeCunn, Hierarchical constrained automatic learning neural network for character recognition, *Google Patents* (1991), p. 19