Computing and Informatics, Vol. 41, 2022, 1337–1357, doi: 10.31577/cai_2022_5_1337

CHARACTER AND WORD EMBEDDINGS FOR PHISHING EMAIL DETECTION

Nikola Stevanović

Faculty of Sciences and Mathematics, University of Niš
Višegradska 33, 18000 Niš, Serbia
e-mail: nikola.stevanovic@pmf.edu.rs

Abstract. Phishing attacks are among the most common malicious activities on the Internet. During a phishing attack, cybercriminals present themselves as a trusted organization or individual. Their goal is to lure people to enter their private information, such as passwords and bank card numbers, while believing that nothing malicious is happening. The attack often starts with a phishing email, which is an email that is very similar to a legitimate email, but usually contains links to malicious websites or uses some other techniques to mislead victims. To prevent phishing attacks, it is crucial to detect phishing emails and remove them from email inbox folders. In this paper, a neural network based phishing email detection model is proposed. In comparison to some earlier approaches, our model does not use manually engineered input features. It learns character and word embeddings directly from email texts, and uses them to extract local and global features using convolutional and recurrent layers, respectively. Our model is tested on the two commonly used datasets for phishing email detection, the SpamAssassin Public Corpus and Nazario Phishing Corpus, and it achieves an accuracy of 99.81 % and F_1-score of 99.74 %, which is on par or better than the current state-of-the-art approaches.

Keywords: Cybersecurity, deep learning, phishing attack, phishing email detection, word embedding

Mathematics Subject Classification 2010: 68-T99

1 INTRODUCTION

Nowadays, online services play a crucial role in our everyday lives. Whether we want to order some goods, entertain ourselves or communicate with our friends and co-workers, we do so over the Internet. Electronic mail (email) is a frequently used way of communication over the Internet, especially in professional settings. According to the Radicati Group[1], the total number of business and consumer emails sent and received per day is expected to reach 319.6 billion by the end of 2021. They predict that the number of worldwide email users will be over 4.3 billion by that time. It is essential to have safe and reliable email communication.

One of the main threats in contemporary cybersecurity are phishing attacks. An attack often starts with a phishing email, which looks like a legitimate email (ham email), but usually contains links to phishing websites. The goal of the attackers is to establish trust from the people who receive the email, and for them to believe that nothing malicious is happening and to click on malicious links. When a person arrives on a phishing website, the person is asked to enter some sensitive information. It can be anything which can later generate some profit to the attackers, including passwords, bank card numbers, confidential information and similar. Phishing websites look like websites of trusted organizations, and victims enter their information believing that they are using original websites of those trusted organizations. Sometimes, attackers impersonate a trusted friend or a co-worker, and directly send emails asking victims for some benefits.

According to the Anti-Phishing Working Group[2], the number of phishing attacks doubled over the course of the year. The same report also states that the number of phishing websites which use SSL/TLS increased to 84 %. As phishing attacks have become more frequent, research in the field of their detection has become increasingly important. Depending on the content that is being analyzed, there are several subfields in the domain. Some authors try to prevent phishing attacks by detecting phishing URLs (Uniform Resource Locator) [3, 4, 5]. In addition to the URL, Feng et al. [6] also used the content of a page and its DOM to predict if it is a phishing webpage. Mehanović et al. [7] applied feature selection on manually engineered website features and used some classical machine learning algorithms for classification. As detecting phishing attacks from the moment people receive phishing emails can drastically mitigate the danger, the focus of this paper is on preventing phishing attacks by detecting phishing emails.

Most of the current literature focuses on selecting the most useful input features from emails, and then applying some classical machine learning algorithms. There

[1] The Radicati Group Inc. Email Statistics Report, 2017-2021. Available at: `https://www.radicati.com/wp/wp-content/uploads/2017/01/Email-Statistics-Report-2017-2021-Executive-Summary.pdf`, 2017.

[2] The Anti-Phishing Working Group. Phishing Activity Trends Report, 4th Quarter 2020. Available at: `https://docs.apwg.org/reports/apwg_trends_report_q4_2020.pdf`, 2021.

is a lack of approaches which learn features directly from phishing emails. In this paper, an approach which learns input features automatically by learning character and word embeddings from email texts is presented. This approach is more general, and it is easier to later update the model with new types of emails.

We propose a neural network classifier, which simultaneously uses character and word embeddings. It combines several convolutional and recurrent layers to extract both local and global features. To evaluate our approach, we downloaded two commonly used datasets in this field, the SpamAssassin Public Corpus [1] and the Nazario Phishing Corpus [2]. The datasets contain emails which are in raw format, so we first created a script to extract only their textual content. To prove the effectiveness of our classifier, we conducted a detailed testing and analysis using the extracted data from those datasets. Experimental results show that our approach is as good or better than the best available approaches.

Traditionally, blacklists were used to detect phishing attacks. This approach was not able to detect modifications of existing phishing attacks and newly created phishing attacks. To overcome this problem, researchers started applying machine learning algorithms. Shaukat et al. [8] presented a survey on application of machine learning algorithms for cybersecurity in the last decade.

In [9] Islam and Abawajy presented a 3-tier model for classification between phishing and legitimate emails. They used 21 features extracted from the email body and header, and a different well-known machine learning algorithm in each tier. If the first two tier algorithms predict the same label, that label is the final prediction of the model. Otherwise, the third tier algorithm decides the model prediction. They achieved an accuracy rate of 97 %. Fette et al. [10] used only 10 input features, 7 binary and 3 continuous features. The continuous features are the number of links and domains, and the maximum number of dots in any of the links present in the email. The binary features include the presence of IP-based links, recently registered domain names, HTML elements and similar. The method uses a random forest classifier with 10 decision trees, and detects over 96 % of the phishing emails. Gualberto et al. [11] first created a Document-Term Matrix, which represents each email in a row, and each term in a column. Then they used Latent Dirichlet Allocation to reduce dimensionality, and employed Extreme Gradient Boosting (XGBoost) classifier. They obtained accuracies of 99.95 % and 99.58 % with and without using knowledge from the Wordnet [12] for lemmatization, respectively.

Gangavarapu and Jaidhar [13] proposed a bio-inspired hybrid metaheuristic to obtain an informative feature subset from 40 content- and behavior-based features and 200 Doc2Vec features. They used a multi-layer perceptron for classification of unsolicited bulk emails (UBEs), and achieved an accuracy of 99.4 % when classifying between phishing and legitimate emails. By utilizing fastText[3] to represent texts as vectors, Ganesh et al. [14] created a phishing email classifier which obtained an F_1-score of 99 %. Yasin and Abuhasan [15] used an intelligent preprocessing phase

[3] FastText. Available at: `https://fasttext.cc/`, accessed on 18. 12. 2021.

to extract a set of 16 features suitable for the phishing detection problem. After evaluating their approach using a 10-fold cross-validation technique, they achieved an accuracy rate of 99.1 % with a random forest classifier. Gangavarapu et al. [16] performed an extensive analysis in which they compared several different feature extraction and machine learning methods for classifying unsolicited bulk emails. By selecting only 21 features from the original set of 40 features, and using a random forest classifier, they achieved an accuracy rate of 99.4 %. Akinyelu and Adewumi [17] extracted a set of 15 phishing features and employed a random forest classifier. They attained an accuracy rate of 99.7 % on a dataset which contains phishing and ham emails in the ratio of 1:9.

By using the skip-gram architecture for Word2Vec to learn word embeddings, Vinayakumar et al. [18] utilized LSTM to obtain an accuracy rate of 99.1 % in a 10-fold cross-validation evaluation. Fang et al. [19] proposed a neural network approach to phishing email detection, based on a recurrent convolutional neural network [20]. They also used Word2Vec to learn character and word embeddings. They attained an accuracy rate of 99.848 % and an F_1-score of 99.331 %. Moradpoor et al. [21] used Word2Vec embeddings and 4 additional features as input to their neural network classifier. They achieved an accuracy rate of 91.5 % on the testing dataset.

Halgaš et al. [22] proposed a recurrent neural network based approach to phishing email detection. They selected 5000 tokens (including special tokens) from email texts, gave each token a unique ID, and presented each email as a sequence of token IDs. Each token ID is being substituted with its embedding vector of size 200 before being used as the input of the recurrent neural network. All the embedding vectors are learned together with other parameters of the model. This method achieved an accuracy rate of 98.91 % and F_1-score of 98.63 %. By learning word embeddings together with other parameter of a convolutional neural network, Hiransha et al. [23] obtained accuracies of 94.2 % and 96.8 % when classifying emails with and without their headers, respectively. Papers [22] and [23] are the most similar to our approach, as they both learn embeddings simultaneously with other neural network parameters.

The remainder of the paper is organized as follows. Theoretical background of all the building blocks used in our model is given in Section 2. In Section 3, our proposed architecture for phishing email detection is described, as well as the datasets that are used for evaluation. Explanation of the evaluation procedure, implementation details and evaluation results are also given in that section. Section 4 contains final comments and some ideas for further improvements.

2 THEORETICAL BACKGROUND

Our approach is a neural network based classifier, which contains multiple building blocks. In this section, we give a theoretical background of different types of building blocks that are used. We also explain what is the function of each of them in our

architecture. A description of how they are interconnected is given in Sections 3.2 and 3.3.

2.1 Embeddings

Embedding layers are used when there is a dictionary of tokens, and there is a need for each token to have its vector representation. Each token has its unique ID in the dictionary (an integer in the range between 0 and $N-1$, where N is the size of the dictionary). An embedding layer has a single matrix of weights $E \in \mathbb{R}^{N \times d}$, where d represents the embedding size. Let us label with e_i the i^{th} row of the matrix $E = [e_0, \ldots, e_{N-1}]^\intercal$. Vector e_i is a vector representation of the token whose ID in the dictionary is i.

On its input, the layer receives a list of L integers $i_0, \ldots, i_{L-1}$, which represent IDs of the tokens from an input sequence. We denote with L the input sequence length. The layer then selects appropriate rows from the embedding matrix E, and ends up with a matrix $[e_{i_0}, \ldots, e_{i_{L-1}}]^\intercal$, which is the output of the embedding layer.

In our work, input samples are emails which are in textual format. As neural networks expect numerical input, we use embedding layers to transform email texts to sequences of token embeddings. We learn embeddings of two types of tokens: characters and words. For each type of token, we learn two embedding layers, one for recurrent and one for convolutional processing. We decided to use separate embedding layers for recurrent and convolutional processing to give the model more flexibility. There are a total of 4 embedding layers in our architecture.

One important advantage of using embedding layers is that they can be learned together with all other parameters of a model, by using gradient based optimization strategies. In our architecture, when a token appears in an email, its two embedding vectors (for recurrent and convolutional processing) are selected and used in the forward pass (together with embedding vectors of all the other tokens from the email). The backpropagation technique is then used to calculate the gradients of the binary cross-entropy loss function with respect to all the network parameters, including the selected embedding vectors. Those gradients are then used by a gradient based optimization algorithm to adjust the selected embedding vectors (as well as all the other network parameters).

2.2 Linear Layer and Nonlinearities

The linear layer is one of the most used layers in neural network architectures, and has two parameters: a matrix $W \in \mathbb{R}^{d_2 \times d_1}$ and a vector $b \in \mathbb{R}^{d_2}$. The input is a d_1-dimensional vector x, and the output of the layer is a d_2-dimensional vector $v = Wx + b$.

Nonlinearities are essential to make more complex representations from the input data. In this paper, three of the standard nonlinearities are used: sigmoid (σ), hyperbolic tangent (tanh) and rectified linear unit (ReLU). Their formulas are

given in Equations (1), (2) and (3). All the nonlinearities are applied element-wise, regardless of the input shape.

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}, \tag{1}$$

$$\tanh(x) = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}}, \tag{2}$$

$$\mathrm{ReLU}(x) = \max(0, x). \tag{3}$$

2.3 Long Short-Term Memory

Long short-term memory [24], or LSTM in short, is a recurrent cell architecture which does not have as much problem with vanishing gradients as some traditional recurrent cells. Recurrent cells are used for processing sequential data. Let us assume that there is a sequence of L d-dimensional input vectors $x_0, \ldots, x_{L-1}$. An LSTM cell sequentially receives input vectors x_t one by one (from x_0 to x_{L-1}), and uses them to update its two d-dimensional state vectors: a cell state c_t and a hidden state h_t. The initial cell and hidden states are usually initialized to zero vectors. The formulas for updating are given in Equations (4), (5), (6), (7), (8) and (9).

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}), \tag{4}$$

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}), \tag{5}$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}), \tag{6}$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}), \tag{7}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t, \tag{8}$$

$$h_t = o_t \odot \tanh(c_t). \tag{9}$$

Symbol $\odot$ represents element-wise multiplication. The cell uses gating mechanisms to control information flow. The input gate i_t controls the amount of information that the cell receives from its input. The forget gate f_t is responsible for resetting the knowledge that is already present in the cell state. The output gate o_t controls how much of the information that exists in the cell should be forwarded to the cell output. A graphical illustration of an LSTM cell is given in Figure 1.

The cell has eight matrix parameters ($W_{ii}, W_{if}, W_{io}, W_{ig}, W_{hi}, W_{hf}, W_{ho}, W_{hg} \in \mathbb{R}^{d\times d}$), and eight vector parameters ($b_{ii}, b_{if}, b_{io}, b_{ig}, b_{hi}, b_{hf}, b_{ho}, b_{hg} \in \mathbb{R}^{d}$). These parameters are being adjusted during the training phase, using a gradient based optimization strategy. The last hidden state of the cell, h_{L-1}, is often used as a representation of the whole input sequence.

There is also a bidirectional variant of the LSTM architecture. It uses two separate LSTM cells. One cell processes input vectors x_t in the correct order (from x_0

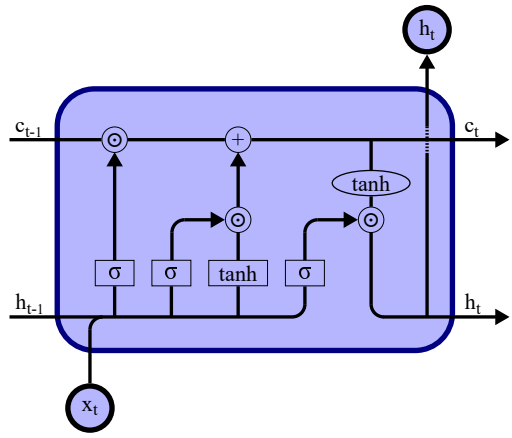

Figure 1. LSTM cell

to x_{L-1}), and the other processes them in the opposite order (from x_{L-1} to x_0). The last hidden states of both directions are then concatenated, and it is a representation of the whole input sequence.

In our architecture, we use two bidirectional LSTM cells, one for processing a sequence of characters and one for processing a sequence of words from each email. Those cells process whole email sequences, and extract global features. Those features are later combined with local features extracted by convolutional layers.

2.4 1-Dimensional Convolution

In the last few decades, convolutional layers have been successfully applied in various fields [25, 26, 27, 28]. For extraction of features in textual data, the most used convolutional layer is the 1-dimensional convolutional layer. Let us assume that there is a matrix $U \in \mathbb{R}^{L \times d}$ formed from a sequence of L d-dimensional vectors on the input of the layer. The layer has multiple kernels, and each of them has a separate matrix of weights $W \in \mathbb{R}^{S \times d}$ and bias parameter $b \in \mathbb{R}$. Hyperparameter S is called the kernel size. Output of the convolution using this kernel is a vector $v \in \mathbb{R}^{L-S+1}$, whose values can be calculated using the formula in Equation (10). The calculation is visually illustrated in Figure 2.

$$v_i = b + \sum_{j=0}^{S-1} \sum_{k=0}^{d-1} W_{j,k} U_{i+j,k}. \tag{10}$$

If the convolutional layer has N kernels, each of them will produce a separate $(L - S + 1)$-dimensional vector on its output. By concatenating them, we get the output of the convolutional layer, which is a $(L - S + 1) \times N$ matrix. The number of rows in the output matrix can additionally be increased by using padding. A certain

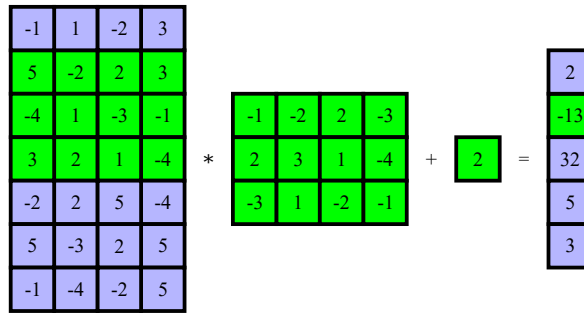

Figure 2. 1-dimensional convolution

number of rows (usually filled with zeros) is added to the input matrix U at its beginning and end, and consequently the size of the output matrix increases for the same number of rows.

Convolutional layers extract local features, with kernel size S determining how many consecutive input sequence elements should be used for feature extraction. For extraction of more diverse local features, we apply multiple convolutional layers with different kernel sizes on sequences of both character and word embeddings in our architecture.

2.5 Global Max-Pooling

The global max-pooling layer is used to calculate strongest activations of the input along its temporal dimension (sequence length). The input is a matrix $U \in \mathbb{R}^{L \times d}$, and the output of the layer is a d-dimensional vector v whose elements can be calculated as

$$v_j = \max_i U_{i,j}. \tag{11}$$

In phishing email detection, finding a malicious part of text somewhere in an email is a good indicator of it being a phishing email. Since a convolutional layer extracts features from local parts of email texts, applying the global max-pooling layer on its output can help us to find places in text where the strongest activations of malicious patterns occur. That makes this method of aggregation over temporal dimension particularly useful in this domain.

2.6 Layer Normalization

By normalizing activities of neurons, it is possible to have faster and more efficient training. Contrary to some earlier normalization techniques, like batch normalization [29], layer normalization [30] calculates the mean and variance values used for normalization on each training sample individually. It performs exactly the same

computation during training and testing, which is not the case with batch normalization.

Let us assume that there is an input vector $x \in \mathbb{R}^d$, which is a representation of a single training sample. The layer has two learnable parameters, γ, $\beta \in \mathbb{R}^d$. Before starting the training phase, γ is initialized to a vector of ones, and β to a vector of zeros. Output of the layer is a d-dimensional vector v, which can be calculated as

$$v = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} \odot \gamma + \beta. \tag{12}$$

Since x is a vector, and $\mathrm{E}[x]$ is a scalar, $\mathrm{E}[x]$ is subtracted from each element of x. The mean and variance values are calculated as shown in Equations (13) and (14), respectively. The variance is calculated using the biased estimator. Hyperparameter ϵ is a small constant (e.g. 10^{-5}), which is used to prevent potential division by zero.

$$\mathrm{E}[x] = \frac{1}{d} \sum_{i=0}^{d-1} x_i, \tag{13}$$

$$\mathrm{Var}[x] = \frac{1}{d} \sum_{i=0}^{d-1} (x_i - \mathrm{E}[x])^2. \tag{14}$$

In addition to making the training process more efficient, this layer also has a positive influence on model generalization. In our architecture, we apply this layer before concatenating representations created using recurrent and convolutional processing, and also before concatenating representations created using characters and words.

3 PROPOSED APPROACH

3.1 Dataset

All the emails used in this research are collected from two datasets, the SpamAssassin Public Corpus [1] and the Nazario Phishing Corpus [2]. The former contains legitimate emails, and the latter contains phishing emails.

From the SpamAssassin Public Corpus emails contained in files whose names end with "ham" are collected. There are three files from the corpus that are used in our analysis: "20030228_easy_ham.tar.bz2", "20030228_hard_ham.tar.bz2" and "20030228_easy_ham_2.tar.bz2". The first file ("easy_ham") comprises legitimate emails which do not contain HTML elements and are relatively easy to classify. The "hard_ham" file contains much trickier legitimate emails, which incorporate HTML elements, spam sounding sentences and similar. The third file comprises legitimate emails which were subsequently added to the dataset. Other files include different or obsolete versions of the same files, or spam emails. Those

files are not included in our analysis. Our final dataset contains 4 150 legitimate emails.

We collected phishing emails from the file "phishing3.mbox" from the Nazario Phishing Corpus. Those are the same emails which were used in [11]. There is a total of 2 279 phishing emails in our final dataset.

Both phishing and legitimate emails are in a raw format in the original datasets. We extract the subject and textual content of the body of each email. Some emails are multipart, and we concatenate the subject and texts from all those parts with new line characters in between. Texts from the files attached to emails are not used. Some parts are in the HTML format, and only textual content and links are extracted from them. From each link, we extract both the displayed text of the link and the URL address that the link directs to. The html2text[4] library is utilized to perform the extraction.

3.2 Character and Word Dictionaries

To analyze emails with our model, we first extract tokens from each email. Two different groups of tokens are used, characters and words. We define a word as a consecutive sequence of letters from an email text that is not part of a larger consecutive sequence of letters. For example, if an email contains text "Hello, Mark", two words will be extracted, "Hello" and "Mark". The same text contains eleven characters: "H", "e", "l", "l", "o", ",", " ", "M", "a", "r" and "k".

We create a dictionary of all tokens for each of the two groups of tokens separately. Only tokens from the training set are used to create the dictionaries. Since token embeddings learned just from a few appearances of a token can be misleading, only tokens which appear in at least ten emails are included. By reducing the total number of observed tokens, we consequently reduce the total number of learnable parameters of the model, making its representation more compact. We found ten to be a good balance between removing rare tokens which can represent noise and be misleading, and keeping other tokens which contain useful information for the classifier.

Both dictionaries contain two additional tokens, one which represents unknown tokens, and one for padding. We classify as unknown all the tokens which appear in less than 10 emails in the training set, including all the tokens which do not appear in the training set (those tokens that the model encounters for the first time during testing). Treating rare tokens as unknown during training improves generalization of the model, because it prepares the model to deal with new rare tokens that can appear during testing. Since a 10-fold cross-validation is used to evaluate our approach, and each of the 10 phases of evaluation has a different training set, different dictionaries are created in each phase.

[4] Html2text. Available at: `https://pypi.org/project/html2text/`, accessed on 18. 12. 2021.

Each token from a dictionary has a unique ID. ID of 0 is reserved for padding tokens, and ID of 1 for unknown tokens. When an email needs to be processed, two sequences of tokens are first extracted, a sequence of characters and a sequence of words. After that, those two sequences are transformed by swapping all the tokens that have their own IDs with those IDs, and all other tokens with 1s. Because our model processes emails in mini-batches, shorter sequences are padded with zeros, so that all the sequences of the same type (characters or words) in one mini-batch have equal length. The result of this process are two 2-dimensional tensors of IDs. The first dimension in each of them represents the batch size, and the second dimension has the size of the sequence of tokens in the mini-batch with the maximal length (in one case the tokens are characters, and in the other words).

3.3 Model Architecture

Our architecture begins processing each of the two tensors of IDs with a block of the same structure but with separate parameters. Each block has 2 embedding layers, one for recurrent and one for convolutional processing (there are 4 embedding layers in the whole architecture). All the embedding matrices have the same number of columns (embedding size). The number of rows is equal to the number of elements in the dictionary (the size depends whether it is an embedding for characters or words). In each of the two blocks, we swap IDs from the tensor with appropriate embedding rows from those matrices, and end up with two new tensors, one for recurrent and one for convolutional processing. Those two tensors are then processed with the remainder of the block. The tensors are 3-dimensional, with the first dimension being the mini-batch size, the second dimension being the maximal token sequence length of all the emails from the mini-batch (depends whether the embeddings layers are from the block that tokenizes based on characters or based on words), and the third dimension being the embedding size (40).

A bidirectional LSTM is used for recurrent processing. The LSTM cell is chosen because it does not have as much problem with vanishing gradients as some traditional recurrent cells. The bidirectional variant of the model gives us an even richer representation of the input sequence. The last hidden states in both directions are concatenated, and that represents a recurrent representation of the block. Since the hidden state size is 40 in our architecture, the recurrent representation of the block is 2-dimensional, with the first dimension being the mini-batch size and the second dimension being of size 80 (two times the size of the hidden state).

Convolutional processing is based on Kim's paper [26]. It starts with four 1-dimensional convolutional layers with different kernel sizes (1, 3, 5 and 7). Each of them has 20 kernels, and we apply padding on both sides of the input for the same amount as the size of the kernels (we increase the second dimension of the 3-dimensional input with padding). The kernel sizes and number of kernels are empirically determined to maximize classification performance. Convolution is applied on the temporal dimension (along the sequence length). After that, we apply rectified linear unit (ReLU) nonlinearity and global max-pooling

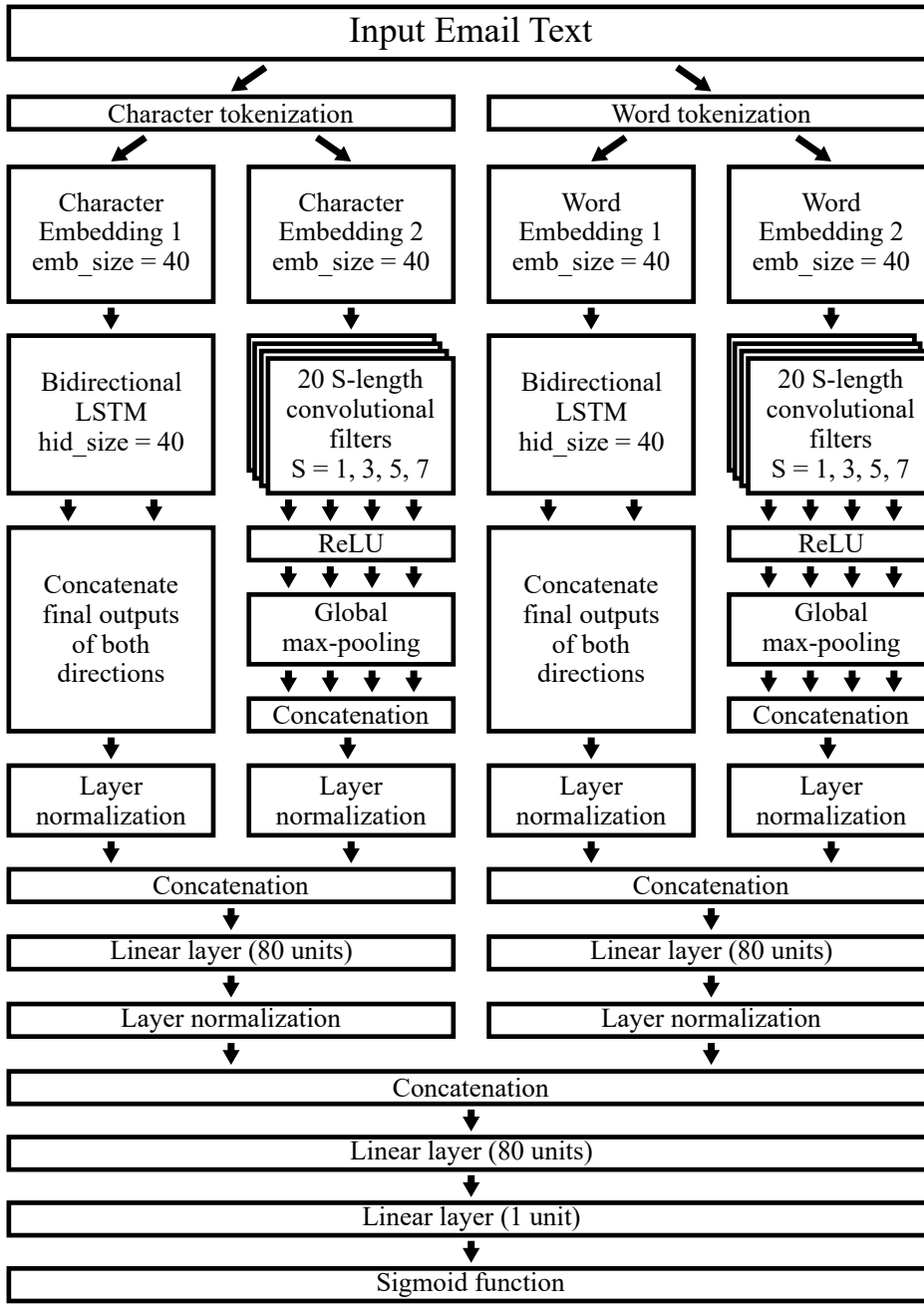

Figure 3. Our proposed architecture

on the output of each of the four convolutional layers. Since shorter token sequences in a mini-batch are padded with zeros, by applying padding in convolutional layers we ensure that the output of global max-pooling for an email is always the same, regardless of the lengths of the token sequences of other emails from the mini-batch. Global max-pooling is used to calculate strongest activations along the temporal dimension (sequence length). Since global max-pooling reduces the temporal dimension, each of the four outputs is 2-dimensional, with the first dimension being the mini-batch size, and the second dimension being the number of kernels (20). To create a convolutional representation of the block, all of them are concatenated in the second dimension (which now has a size of $4 \cdot 20 = 80$).

We apply layer normalization on both representations (recurrent and convolutional) from each of the two blocks (one which analyzes characters and one which analyzes words). Layer normalization is used to improve training efficiency and speed, but also to improve model generalization. To create a representation of a block, we first concatenate its two normalized representations, recurrent and convolutional, and then apply a fully connected linear layer. The linear layer has 160 input neurons and 80 output neurons.

Representations of both blocks, one which processes characters and one which processes words, are normalized with layer normalization and then concatenated. After that, one additional fully connected linear layer is applied, which combines representations of both blocks in a single representation. The layer has 160 input and 80 output neurons. Finally, one final linear layer with a single output node is used. That node uses a sigmoid activation function, and its output represents the probability that it is a phishing email. The whole architecture is depicted in Figure 3.

3.4 Evaluation

To implement our model, we have used the PyTorch [31] library for machine learning. All experiments were carried out on Google Colab[5], using a graphics processing unit (GPU).

Configuration details of our proposed model and training procedure are given in Table 1. A binary cross-entropy loss function is used to train the model. The model parameters are updated by the Adam [32] optimizer, with a learning rate of 0.001. The model is trained for 4 epochs, using a mini-batch size of 32. Mini-batches are created by sorting all emails by their text length, and then putting adjacent emails in the same mini-batch. Putting emails of similar lengths, and consequently of similar number of tokens, in the same mini-batch can drastically speed up the model. Before each epoch, the order in which the model receives mini-batches is randomly changed. It is common in the literature to discard very small or large emails from the dataset, or to cut or extend them to a certain length. While it can be ben-

[5] Google Colab. Available at: `https://colab.research.google.com/`

Layer	Count	Details
Embedding	4	embeddings size: 40
Bidirectional LSTM	2	hidden size: 40
1d convolution	2	20 kernels of size 1 20 kernels of size 3 20 kernels of size 5 20 kernels of size 7
Layer normalization	6	$\epsilon = 10^{-5}$
Linear layer	3	input size: 160, output size: 80
	1	input size: 80, output size: 1
Global max-pooling	2	
Optimization details	optimizer	Adam
	epochs	4
	learning rate	10^{-3}
	batch size	32

Table 1. Configuration details of our model

eficial while testing on the dataset, these classifiers can easily be deceived in real application. We use all emails in their full length.

Fold No.	Ham	Phishing	Fold No.	Ham	Phishing
1	415	227	7	415	228
2	415	228	8	415	228
3	415	228	9	415	228
4	415	228	10	415	228
5	415	228			
6	415	228	Total	4 150	2 279

Table 2. Number of legitimate and phishing emails in each fold

To evaluate our approach, we collected phishing emails from the Nazario Phishing Corpus and legitimate emails from the SpamAssassin Public Corpus, and extracted texts from their subjects and bodies. Because the sizes of those two datasets are relatively small, the often used train-validation-test split could produce unstable results. That is why we decided to use a 10-fold cross-validation to evaluate our model. All emails are randomly divided in 10 groups, maintaining in each group approximately the same ratio between phishing and legitimate emails as the ratio between the sizes of the two original datasets. Table 2 shows the exact number of legitimate and phishing emails in each group. In each of the 10 experiments, emails from one group are used for testing, and emails from the remaining 9 groups are used for training. As a result, each email is used exactly once for testing, and those results are used to calculate all evaluation metrics.

In order to evaluate our proposed phishing email detection model, we applied the following evaluation metrics: accuracy, precision, recall (true positive rate), F_1-

score and false positive rate (FPR). Their formulas are given in Equations (15), (16), (17), (18) and (19), in function of the number of true positive (tp), false negative (fn), false positive (fp) and true negative (tn) samples. Additionally, confusion matrices are used to compare different variations of our approach. We also show a receiver operating characteristic (ROC) curve of our model. The ROC curve plots true positive rate against false positive rate at various threshold values. The whole training and evaluation procedure are summarized by the pseudo-code in Algorithm 1.

$$\text{Accuracy} = \frac{tp + tn}{tp + fn + fp + tn}, \tag{15}$$

$$\text{Precision} = \frac{tp}{tp + fp}, \tag{16}$$

$$\text{Recall} = \frac{tp}{tp + fn}, \tag{17}$$

$$\text{F}_1\text{-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{18}$$

$$\text{FPR} = \frac{fp}{fp + tn}. \tag{19}$$

Model	Accuracy	Precision	Recall	F_1-score	FPR
Full	0.99813	0.99824	0.99649	0.99736	0.00096
Only characters	0.99564	0.99429	0.99342	0.99385	0.00313
Only words	0.99362	0.99513	0.98684	0.99097	0.00265

Table 3. Results of our proposed model

Table 3 depicts classification results of our model. The results of the full model are presented, as well as the results of the models which use only characters or only words (containing left or right side of the architecture in Figure 3 up to the concatenation layer which merges the normalized recurrent and convolutional representations, and everything after the last concatenation layer). The table shows that the full model which uses both characters and words achieved better results than the models which use only one type of token. Our full model achieved an accuracy rate of 99.81 % and F_1-score of 99.74 %. The model which works only with characters has shown greater results than the model that uses only words. They achieved accuracy rates of 99.56 % and 99.36 %, respectively. The model which extracts only words has a slightly better precision and false positive rate. The real difference is in the recall scores, which is considerably higher for the model that uses only characters. Our full model achieved better results than the other two variants of the model on all five evaluation metrics.

Algorithm 1: Training and evaluation procedure

Data: $F_1 = (X_1, Y_1), \ldots, F_{10} = (X_{10}, Y_{10})$; `// dataset folds`
for $testInd \leftarrow 1$ **to** 10 **do**
  $model$ = initializeModel();
  $optimizer$ = initializeAdamOptimizer($model.weights$);
  $D_{train} = \bigcup_{i=1, i \neq testInd}^{10} F_i$;
  $D_{test} = F_{testInd}$;
  $Dict_{char}, Dict_{word}$ = createDictionaries(D_{train});
  $miniBatches$ = chunked(sort(D_{train})); `// sort by email text length and divide in chunks of size 32`
  for $epoch \leftarrow 1$ **to** 4 **do**
    shuffle($miniBatches$);
    foreach $miniBatch=(X_{mb},Y_{mb})$ **of** $miniBatches$ **do**
      $charTokens$ = tokenize($X_{mb}, Dict_{char}$);
      $wordTokens$ = tokenize($X_{mb}, Dict_{word}$);
      out = forwardPass($model, charTokens, wordTokens$);
      $loss$ = binaryCrossEntropyLoss(out, Y_{mb});
      $grads = \frac{\partial loss}{\partial model.weights}$; `// calculated using backpropagation`
      $optimizer$.updateModelWeights($grads$);
    end
  end
  $P_{testInd}$ = makePredictions($model, X_{testInd}$)
end
calculateEvaluationMetrics($Y_1, \ldots, Y_{10}; P_1, \ldots, P_{10}$);

Table 4 shows confusion matrices of our full model and the models that use only one type of tokens. Our full model misclassified only 4 legitimate and 8 phishing emails. This is the variant of our model with the lowest number of both false positives and false negatives. The variant which uses only words has a slightly lower number of false positives than the variant that uses only characters, but the latter has two times less false negatives than the former. Figure 4 shows the ROC curve of our full model. Since it is hard to visually distinguish between the ROC curves of the three variants of the model, we only show the ROC curve of the full variant in the figure. It is clear from the figure that the model successfully separates between phishing and legitimate emails.

In Table 5 our approach is compared with the state-of-the-art models in the phishing email detection literature. Two of the models [10, 11] have variants that additionally extract knowledge from external sources, such as classification results from an external spam filter or the WordNet library. To make a fair comparison, we show in the table results of the variants of those models which do not use external knowledge. Since the accuracy and F_1-score were not given in [10], we calculated them from the false positive rate and false negative rate values given in the paper.

(a) Full model

		Predicted	
		Ham	Phishing
Actual	Ham	4 146	4
	Phishing	8	2 271

(b) Only characters

		Predicted	
		Ham	Phishing
Actual	Ham	4 137	13
	Phishing	15	2 264

(c) Only words

		Predicted	
		Ham	Phishing
Actual	Ham	4 139	11
	Phishing	30	2 249

Table 4. Confusion matrices of different variations of our proposed model

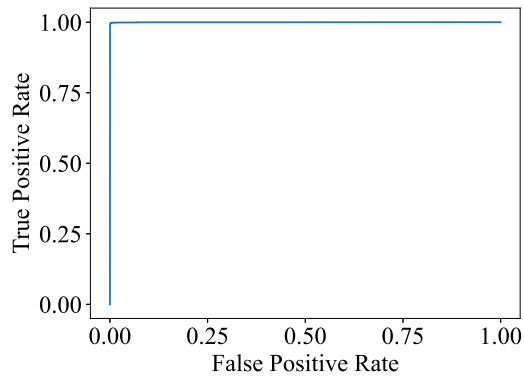

Figure 4. ROC curve of our model

Accuracies and F_1-scores of all the other models that are used for comparison are in the table, except for the F_1-scores of the models [9] and [13], which we could not find.

As can be seen from the table, no model has a higher F_1-score than our model. The second best F_1-score of 99.58 % was achieved by Gualberto et al. [11]. They also have a lower accuracy than our approach. Fang et al. [19] is the only approach that obtained a slightly higher accuracy, but they tested on a very imbalanced dataset, which shows in the F_1-score that they obtained, which is considerably lower than ours. All the other approaches have lower both accuracy and F_1-score than our model.

The two approaches which are the most similar to ours are Halgaš et al. [22] and Hiransha et al. [23]. They are both neural network based approaches which learn their embeddings together with all the other parameters (weights). These two mod-

Reference	Accuracy (%)	F_1-score (%)
Our approach	99.81	99.74
Fang et al. [19]	99.85	99.33
Akinyelu et al. [17]	99.7	98.45
Gualberto et al. [11]	99.58	99.58
Gangavarapu et al. [13]	99.4	N/A
Gangavarapu et al. [16]	99.4	99.4
Yasin et al. [15]	99.1	99.1
Halgaš et al. [22]	98.91	98.63
Fette et al. [10]	98.87	94.68
Islam et al. [9]	97	N/A
Moradpoor et al. [21]	91.5	92.06

Table 5. Comparison with other approaches

els obtained accuracies of 98.91 % and 96.8 %, respectively. With an accuracy rate of 99.81 %, our model represents a considerable improvement over their approaches.

4 CONCLUSION AND FUTURE WORK

In this paper, the problem of phishing email detection has been considered. We first extracted textual content from legitimate and phishing emails. We then utilized character and word embeddings to learn vectorized representations from textual inputs, and proposed a neural network based classifier. We conducted an extensive evaluation of our approach, and confirmed its efficiency.

Instead of manually engineering input features, we extract characters and words from emails. This approach is more general, and it will be much easier to apply it on new types of phishing emails which may appear in the future. We learn character and word embeddings together with other parameters of the model, and achieve better results than other similar approaches. Our model has also shown similar or better performance than the current state-of-the-art models for phishing email detection.

In the future, collecting a larger dataset for phishing email detection would be of great importance to the field. Our model is a neural network with a lot of parameters. As neural networks usually work better with larger datasets, we expect that our model would be able to learn successfully from such datasets.

It is also important to establish a consensus on what parts of a raw email should be used as input features. Including features such as date when the email was sent or the email address of the person who received the email could work better on the dataset than in real application, if the legitimate and phishing emails from the dataset were collected in different time periods or from different groups of people. In this study, only textual data from email subjects and bodies was used.

As the number of phishing attacks constantly increases, more attention should be focused on them. Phishing emails are often the first step in phishing attacks, and

their detection could prevent future harm. We have shown the usefulness of character and word embeddings as input features of neural network based classification models for phishing email detection, and we expect to see more papers with similar approaches in the future.

REFERENCES

[1] Mason, J.: The Apache SpamAssassin Public Corpus. Available at: `https://spamassassin.apache.org/old/publiccorpus/`, 2005.

[2] Nazario, J.: Phishing Corpus. Available at: `https://monkey.org/~jose/phishing/`, 2007.

[3] Christou, O.—Pitropakis, N.—Papadopoulos, P.—McKeown, S.—Buchanan, W.: Phishing URL Detection Through Top-Level Domain Analysis: A Descriptive Approach. Proceedings of the 6th International Conference on Information Systems Security and Privacy – ICISSP, 2020, pp. 289–298, doi: 10.5220/0008902202890298.

[4] Sahingoz, O. K.—Buber, E.—Demir, O.—Diri, B.: Machine Learning Based Phishing Detection from URLs. Expert Systems with Applications, Vol. 117, 2019, pp. 345–357, doi: 10.1016/j.eswa.2018.09.029.

[5] Marchal, S.—Francois, J.—State, R.—Engel, T.: PhishStorm: Detecting Phishing with Streaming Analytics. IEEE Transactions on Network and Service Management, Vol. 11, 2014, No. 4, pp. 458–471, doi: 10.1109/TNSM.2014.2377295.

[6] Feng, J.—Zou, L.—Ye, O.—Han, J.: Web2Vec: Phishing Webpage Detection Method Based on Multidimensional Features Driven by Deep Learning. IEEE Access, Vol. 8, 2020, pp. 221214–221224, doi: 10.1109/ACCESS.2020.3043188.

[7] Mehanović, D.—Kevrić, J.: Phishing Website Detection Using Machine Learning Classifiers Optimized by Feature Selection. Traitement du Signal, Vol. 37, 2020, No. 4, pp. 563–569, doi: 10.18280/ts.370403.

[8] Shaukat, K.—Luo, S.—Varadharajan, V.—Hameed, I. A.—Xu, M.: A Survey on Machine Learning Techniques for Cyber Security in the Last Decade. IEEE Access, Vol. 8, 2020, pp. 222310–222354, doi: 10.1109/ACCESS.2020.3041951.

[9] Islam, R.—Abawajy, J.: A Multi-Tier Phishing Detection and Filtering Approach. Journal of Network and Computer Applications, Vol. 36, 2013, No. 1, pp. 324–335, doi: 10.1016/j.jnca.2012.05.009.

[10] Fette, I.—Sadeh, N.—Tomasic, A.: Learning to Detect Phishing Emails. Proceedings of the 16th International Conference on World Wide Web (WWW '07), 2007, pp. 649–656, doi: 10.1145/1242572.1242660.

[11] Gualberto, E. S.—De Sousa, R. T.—De B. Vieira, T. P.—Da Costa, J. P. C. L.—Duque, C. G.: From Feature Engineering and Topics Models to Enhanced Prediction Rates in Phishing Detection. IEEE Access, Vol. 8, 2020, pp. 76368–76385, doi: 10.1109/ACCESS.2020.2989126.

[12] Miller, G. A.: WordNet: A Lexical Database for English. Communications of the ACM, Vol. 38, 1995, No. 11, pp. 39–41, doi: 10.1145/219717.219748.

[13] Gangavarapu, T.—Jaidhar, C. D.: A Novel Bio-Inspired Hybrid Metaheuristic for Unsolicited Bulk Email Detection. In: Krzhizhanovskaya, V. V. et al. (Eds.): Computational Science – ICCS 2020. Springer, Cham, Lecture Notes in Computer Science, Vol. 12139, 2020, pp. 240–254, doi: 10.1007/978-3-030-50420-5_18.

[14] Ganesh, H. B. B.—Vinayakumar, R.—Kumar, M. A.—Soman, K. P.: Distributed Representation Using Target Classes: Bag of Tricks for Security and Privacy Analytics. In: Verma, R. M., Das, A. (Eds.): Anti-Phishing Shared Task Pilot at the 4th ACM IWSPA (IWSPA-AP 2018). CEUR Workshop Proceedings, Vol. 2124, 2018, pp. 10–15.

[15] Yasin, A.—Abuhasan, A.: An Intelligent Classification Model for Phishing Email Detection. International Journal of Network Security and Its Applications, Vol. 8, 2016, No. 4, pp. 55–72, doi: 10.5121/ijnsa.2016.8405.

[16] Gangavarapu, T.—Jaidhar, C. D.—Chanduka, B.: Applicability of Machine Learning in Spam and Phishing Email Filtering: Review and Approaches. Artificial Intelligence Review, Vol. 53, 2020, No. 7, pp. 5019–5081, doi: 10.1007/s10462-020-09814-9.

[17] Akinyelu, A. A.—Adewumi, A. O.: Classification of Phishing Email Using Random Forest Machine Learning Technique. Journal of Applied Mathematics, Vol. 2014, 2014, Art. No. 425731, doi: 10.1155/2014/425731.

[18] Vinayakumar, R.—Ganesh, H. B. B.—Kumar, M. A.—Soman, K. P.—Poornachandran, P.: DeepAnti-PhishNet: Applying Deep Neural Networks for Phishing Email Detection. In: Verma, R. M., Das, A. (Eds.): Anti-Phishing Shared Task Pilot at the 4th ACM IWSPA (IWSPA-AP 2018). CEUR Workshop Proceedings, Vol. 2124, 2018, pp. 39–49.

[19] Fang, Y.—Zhang, C.—Huang, C.—Liu, L.—Yang, Y.: Phishing Email Detection Using Improved RCNN Model with Multilevel Vectors and Attention Mechanism. IEEE Access, Vol. 7, 2019, pp. 56329–56340, doi: 10.1109/access.2019.2913705.

[20] Lai, S.—Xu, L.—Liu, K.—Zhao, J.: Recurrent Convolutional Neural Networks for Text Classification. Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015, pp. 2267–2273, doi: 10.1609/aaai.v29i1.9513.

[21] Moradpoor, N.—Clavie, B.—Buchanan, B.: Employing Machine Learning Techniques for Detection and Classification of Phishing Emails. IEEE 2017 Computing Conference, 2017, pp. 149–156, doi: 10.1109/SAI.2017.8252096.

[22] Halgaš, L.—Agrafiotis, I.—Nurse, J. R. C.: Catching the Phish: Detecting Phishing Attacks Using Recurrent Neural Networks (RNNs). In: You, I. (Ed.): Information Security Applications (WISA 2019). Springer, Cham, Lecture Notes in Computer Science, Vol. 11897, 2019, pp. 219–233, doi: 10.1007/978-3-030-39303-8_17.

[23] Hiransha, M.—Unnithan, N. A.—Vinayakumar, R.—Soman, K. P.: Deep Learning Based Phishing E-Mail Detection. In: Verma, R. M., Das, A. (Eds.): Anti-Phishing Shared Task Pilot at the 4th ACM IWSPA (IWSPA-AP 2018). CEUR Workshop Proceedings, Vol. 2124, 2018, pp. 16–20.

[24] Hochreiter, S.—Schmidhuber, J.: Long Short-Term Memory. Neural Computation, Vol. 9, 1997, No. 8, pp. 1735–1780, doi: 10.1162/neco.1997.9.8.1735.

[25] Krizhevsky, A.—Sutskever, I.—Hinton, G. E.: Imagenet Classification with

Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J., Bottou, L., Weinberger, K. Q. (Eds.): Advances in Neural Information Processing Systems 25 (NIPS 2012). Curran Associates, Inc., 2012, pp. 1097–1105.

[26] Kim, Y.: Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), ACL, 2014, pp. 1746–1751, doi: 10.3115/v1/d14-1181.

[27] Acharya, U. R.—Oh, S. L.—Hagiwara, Y.—Tan, J. H.—Adam, M.—Gertych, A.—Tan, R. S.: A Deep Convolutional Neural Network Model to Classify Heartbeats. Computers in Biology and Medicine, Vol. 89, 2017, pp. 389–396, doi: 10.1016/j.compbiomed.2017.08.022.

[28] Gatys, L. A.—Ecker, A. S.—Bethge, M.: Image Style Transfer Using Convolutional Neural Networks. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2414–2423, doi: 10.1109/CVPR.2016.265.

[29] Ioffe, S.—Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: Zhao, P., Blei, D. (Eds.): Proceedings of the 32nd International Conference on Machine Learning (ICML 2015). Proceedings of Machine Learning Research (PMLR), Vol. 37, 2015, pp. 448–456.

[30] Ba, J. L.—Kiros, J. R.—Hinton, G. E.: Layer Normalization. 2016, doi: 10.48550/arXiv.1607.06450.

[31] Paszke, A.—Gross, S.—Massa, F.—Lerer, A.—Bradbury, J. et al.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (Eds.): Advances in Neural Information Processing Systems 32 (NeurIPS 2019). Curran Associates, Inc., 2019, pp. 8026–8037.

[32] Kingma, D. P.—Ba, J. L.: Adam: A Method for Stochastic Optimization. 3rd International Conference on Learning Representations (ICLR 2015), 2015, pp. 1–15, doi: 10.48550/arXiv.1412.6980.

Nikola Stevanović received his B.Sc. degrees in both computer science and mathematics from the University of Niš, Faculty of Sciences and Mathematics, Serbia in 2014. He received his M.Sc. degree in computer science from the same university in 2016, where he is currently pursuing the Ph.D. degree in computer science. His research interests include deep learning and its application in cybersecurity.