Phishing URL Detection using Convolutional and Recurrent Neural Networks

By Daniel DeFlores and Ethan Orevillo

Abstract

This study compares the effectiveness of convolutional neural networks (CNN) and recurrent neural networks (RNN) in detecting phishing URLs. It examines the accuracy, time to process, and number of false positives/negatives on a dataset of over 800,000 URLs for each model. To train the models, each URL was tokenized at the character level before processing. Our results show that for higher accuracy, an RNN should be chosen. However, the processing time for the RNN was nearly four times that of the CNN model, with only slightly better performance. Thus, CNN may be more effective depending on the case.

1 Introduction

With the increasing prevalence of cyber threats, particularly phishing attacks, there is a need for automated systems to identify and flag malicious URLs. The sophistication of phishing techniques demands advanced methods that can adapt to these new threats.

Phishing is one of the most common forms of cybercrime. In a study conducted by the Cybersecurity and Infrastructure Security Agency (CISA) last year, within the first 10 minutes of receiving a malicious email, 84% of employees took the bait by either replying with sensitive information or interacting with a spoofed link or attachment [1]. Our work aims to reduce the number of people who fall victim to these fake links, especially those who are not very well-versed with technology. These people may not even know what phishing is, and are the most susceptible.

As an attempt, we built two models on a dataset of 822,010 URLs, with about 48% of them being phishing URLs. We will thoroughly compare each of the models on accuracy, time to process, and number of false positives/negatives. These metrics will provide insight on which model is more practical given the problem.

2 Related Works

In the realm of phishing URL detection, numerous studies have explored various approaches, with a particular emphasis on neural network-based methods. This section compares and contrasts our work with these studies, highlighting the unique aspects and contributions of our research.

One notable study in this field is by Y. Huang et al., titled "Phishing URL Detection via

CNN and Attention-Based Hierarchical RNN" [2]. This research combined Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) with an attention mechanism, aiming to improve the detection of phishing URLs. Their approach differs from ours in that they integrated both CNN and RNN in a single model, whereas our study distinctively separates these two to evaluate their individual effectiveness.

Another significant contribution is the work of Alshingiti et al., "A Deep Learning-Based Phishing Detection System Using CNN, LSTM, and LSTM-CNN" [3]. This study also employed a combined approach, integrating Long Short-Term Memory (LSTM) networks with CNNs. The comparison between their combined model and our separate CNN and RNN models provides valuable insights into the effectiveness of these architectures in isolation versus in conjunction.

Additionally, the research by Ariyadasa et al., titled "Detecting phishing attacks using a combined model of LSTM and CNN" [4], further reinforces the trend of combining LSTM and CNN for phishing detection. Their work emphasizes the synergistic benefits of combining these two neural network types. Our research stands out by isolating each network type, offering a clear comparison of their individual capabilities in the context of phishing URL detection.

Beyond neural networks, other studies have also explored various machine learning techniques for similar purposes. For example, the use of support vector machines, genetic algorithms, and fuzzy logic in phishing detection illustrates the diversity of approaches within this research domain. These studies, while not directly comparable to our CNN and RNN-focused research, provide a broader context within which our work can be understood and contribute to the collective understanding of effective techniques in cybersecurity.

In summary, while the existing literature primarily focuses on combined models of neural networks, our research uniquely contributes by analyzing the separate performances of CNN and RNN models in phishing URL detection. This distinction allows for a more nuanced understanding of each model's strengths and limitations in this specific application.

3 Methodology

3.1 Dataset and Models

The dataset used for our experiment was one of 822,010 URLs, with 394,982 (about 48%) being phishing links, and the rest legitimate. Phishing links were labeled with 0s, and legitimate links were labeled with 1s.

Since URLs do not have particularly defining features (they are all made up of the same characters), we determined that a CNN or RNN would best fit the data, since they are effective in finding similarities and patterns among datasets.

3.2 Experimentation

To utilize both of these methods, the dataset first had to be tokenized into numerical data so that each algorithm could correctly process the text. Next, we had to ensure that there was consistency in the length of the URLs, so the data was padded to a uniform sequence length. These steps allowed for the “Keras” python neural network algorithms to process the data. From

there, we split the dataset into training, testing, and validation sets using “train_test_split()” from the “scikit-learn” [5] machine learning library for python. The testing set was composed of 60% of the original data, the testing set contained 20%, and the validation set used the remaining 20%.

We trained the CNN model with the “Keras” [6] neural network library for Python. We added several layers to the network. We first added an embedding layer to turn the tokenized URLs into vectors that the network can process; we set the input dimension size to the number of unique tokens + 1, the output dimension size to 50, and the max length to 256 (the length we padded each URL to). We added a 1D convolution layer with 32 filters spanning 3 characters to recognize features, and a max pooling layer to highlight the most relevant of these features. We added a dense layer of 64 neurons to receive, combine, and interpret the features from the previous layers. Here, we used the ReLU activation function to allow for more complex patterns in the data. We also added a dropout layer to help resist overfitting by setting some instances to 0 at a rate of 0.5. The final layer was a dense layer consisting of one neuron with a sigmoid function to classify the URL with 0 or 1. We trained this model for 10 epochs.

We trained our RNN model using the Keras library, like our CNN. The process began with tokenizing URLs and converting them into vectors using an embedding layer, with the input dimension set to the number of unique tokens + 1, output dimension to 50, and max length to 256. The core of the RNN was LSTM layers; the first had 64 units and returned sequences for the subsequent LSTM layer. To prevent overfitting, we included dropout layers with a rate of 0.5 after each LSTM layer. A dense layer of 64 neurons with ReLU activation followed, integrating the features from LSTM layers. The final layer was a dense layer with one neuron using a sigmoid activation for binary classification. This model was trained over 10 epochs, balancing the sequence-processing capability of the RNN with the need for precise classification.

After training the models described above, we further processed the data intro metrics. Accuracy, processing time, and number of false positives and false negatives are the metrics we chose because they are generally the most important when it comes to a binary classifier model. Accuracy and the amount of false positives and false negatives determine how successful a model is, and give insight to how the model is fit to the data. Processing time is crucial to determine how scalable a model is. This is especially important for our idea to scale one of the models into a real life application.

4 Results and Discussions

4.1 Convolutional Neural Network

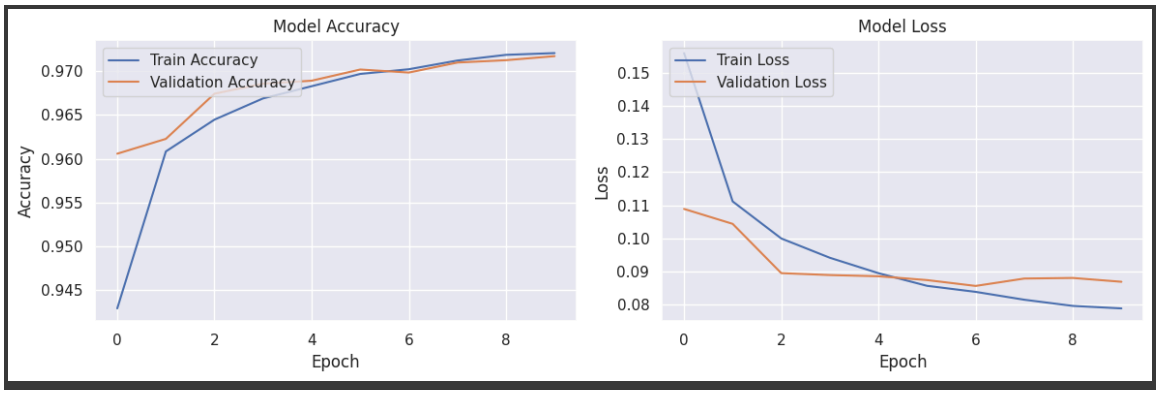

Figure 1: CNN Accuracy and Loss by Epoch

After training the CNN model as explained in section 3.2, the above results were recorded (Figure 1) for accuracy and loss. The accuracy and loss for both the training and validation data were computed after each epoch. After training for 10 epochs, which took just under 17 minutes, the accuracy score for the model was about 0.97. This is a very strong accuracy score, and exceeded our expectations. The processing time was also very quick considering the dataset contained over 800,000 URLs.

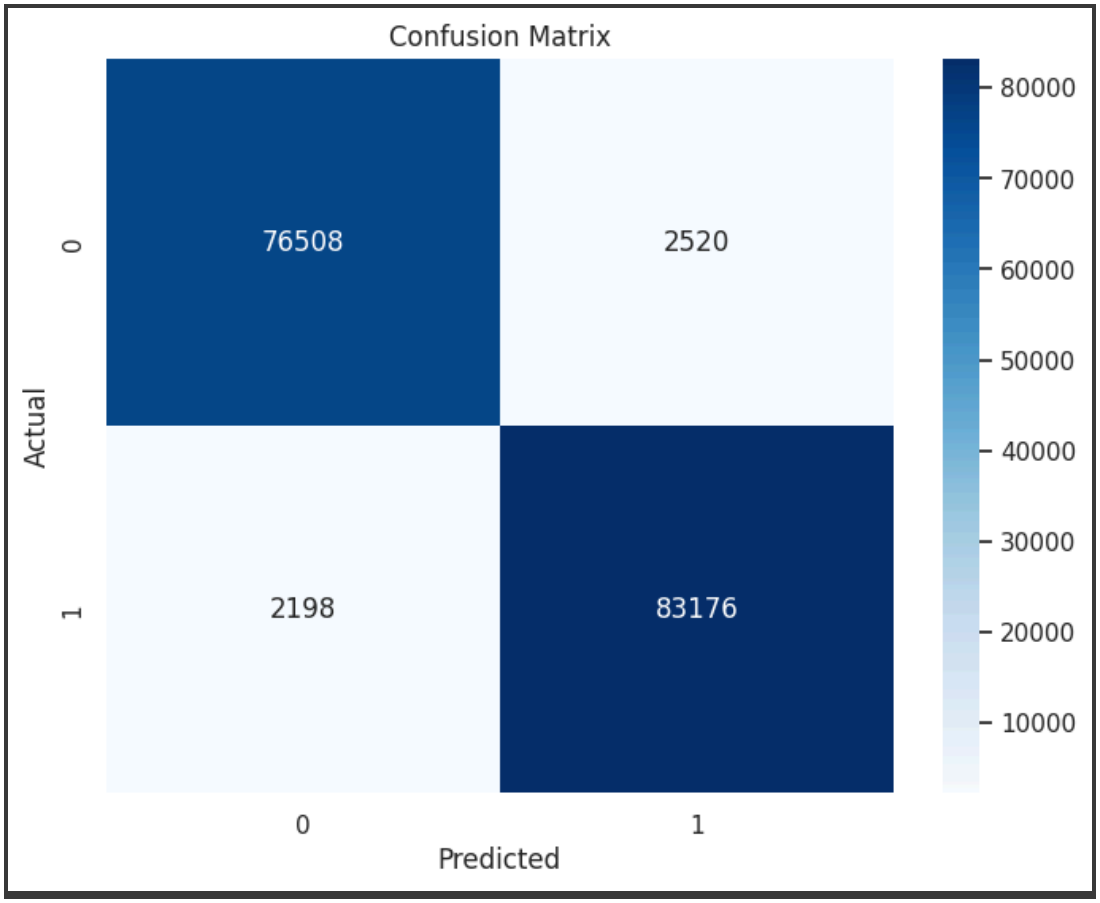

Figure 2: Confusion Matrix of CNN

As shown in Figure 2 above, the number of false positives was 2,520 and the number of false negatives was 2,198. Given that there were 83,176 true positives and 76,508 true negatives, this means that only 3% of the positives were false, and 2.8% of the negatives were false. This indicates a strong performance in correctly classifying both the positive and negative instances of the dataset. The balanced number of false positives and false negatives suggests that our model isn't biased towards either instance. With the values from the confusion matrix we calculated the F1 score as about 0.972. This shows that the model is highly effective at detecting phishing links while minimizing false positives and false negatives.

4.2 Recurrent Neural Network

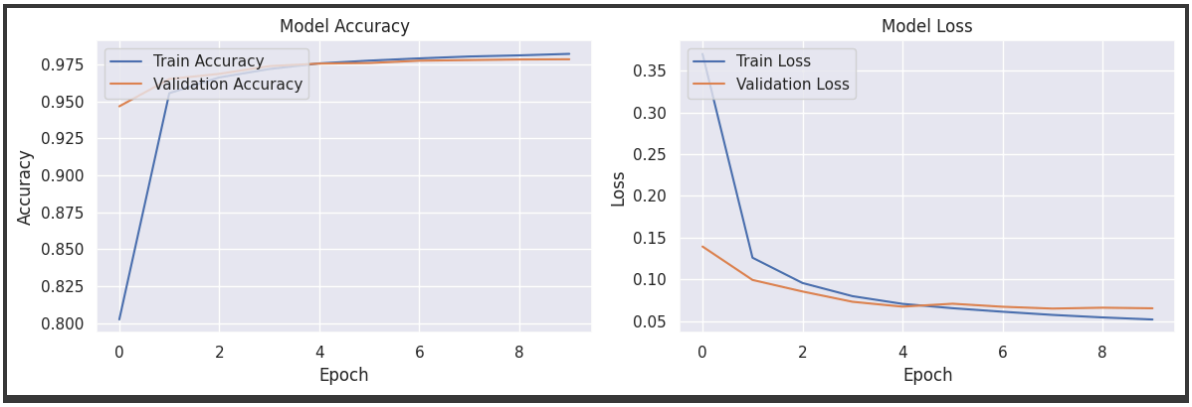

Figure 3: RNN Accuracy and Loss by Epoch

The final results for accuracy and loss on the RNN model are nearly identical to the CNN in section 4.2. Interestingly though, high accuracy was achieved after just a few epochs, and the timeline by epoch seems to be more linear. Notice how straight the lines are as compared to Figure 2, where the lines are more sporadic along the way. Also notice how the accuracy score reaches 0.975 after epoch 4. We theorize that this is because recurrent neural networks process data iteratively, making for a more "smooth" training sequence, and quicker results. The final accuracy for the RNN was about 0.98, making it slightly better than the CNN. However, it took 59 minutes, roughly four times the processing time of CNN. This extra time allocation is a big factor to take into account, and begs the question of whether or not an RNN is worth it.

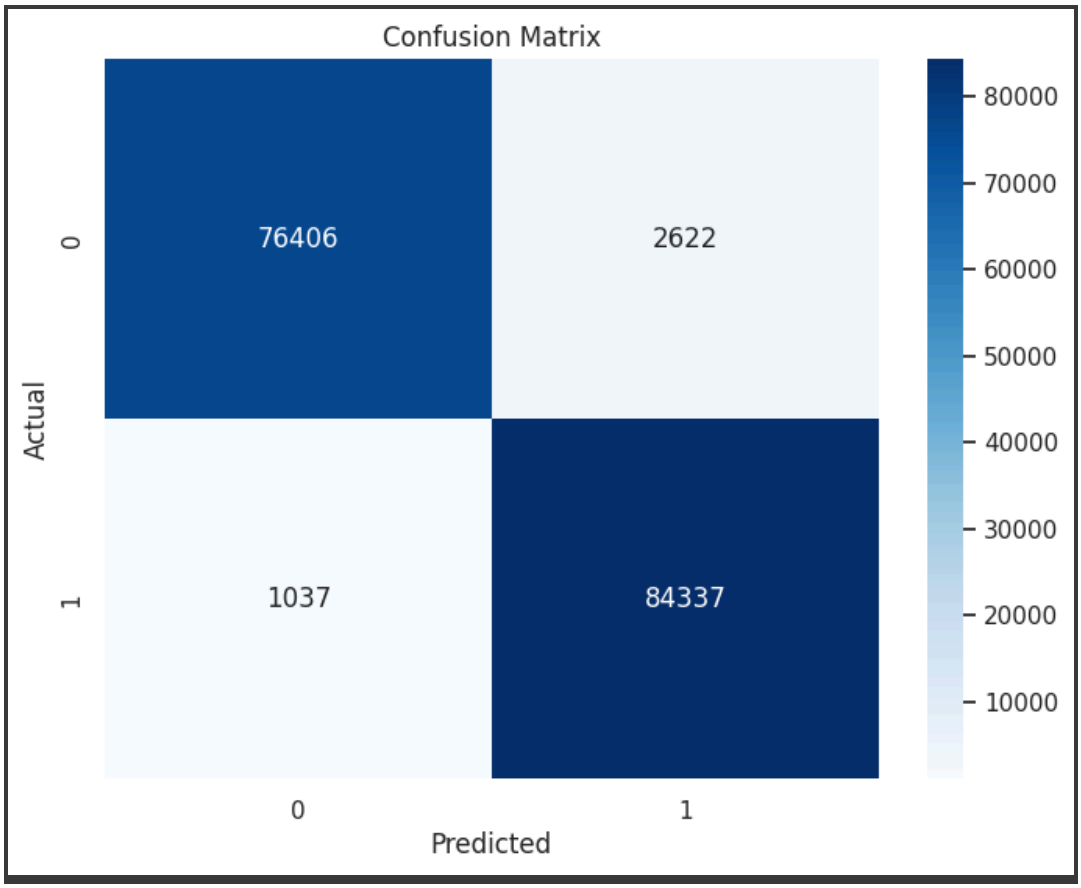

Figure 4: Confusion Matrix of RNN

The number of false positives for the RNN was 2,622, and the number of false negatives was 1,037. The false negatives decreased by over 50% when compared to the CNN results. However, the false positives increased slightly. Given that there were 84,337 true positives and 76,406 true negatives, this means that only about 3.1% of the positives were false, and about 1.4% of the negatives were false. Since the false negatives were cut in half and there was a 0.1% increase in false positives, there may be a slight bias for this model to classify an input as a positive instance. We calculated the F1 score from the values in the confusion matrix to be about 0.979. This is slightly higher than the score of the CNN model, an even better indicator that the model performed more effectively.

5 Conclusion

In this experiment, our objective was to explore and compare how effective convolutional neural networks and recurrent neural networks are in detecting phishing URLs. We trained both a CNN and an RNN on the same dataset of URLs in order to accomplish this. We compared their efficacy based on accuracy, processing time, and number of false positives and false negatives. The CNN model achieved a high accuracy score of 0.97, and the accuracy score of the RNN was 0.98. The processing times for the CNN and RNN were 17 minutes, and 59 minutes, respectively. Additionally, the number of false positives and false negatives were very low for each model. The convolutional network yielded 3% false positives and 2.8% false negatives,

while the recurrent network yielded 3.1% false positives and 1.4% false negatives.

Although the recurrent network performed better than the convolutional network, its processing time was about four times more. We decided that for our intentions, this would not be favorable since scalability is a large priority. The tradeoff is simply not worth it for only slightly better results. In certain rare cases where computational time is not an issue, the RNN model would be best in order to achieve the best results possible in terms of accuracy. On the other hand, for a typical use case a CNN would be much more efficient and the better choice.

We would like to perform further research on RNN models for phishing detection. As we mentioned earlier in section 4.2, the RNN accuracy was better than the CNN after just 4 epochs. In the future, training over 5 epochs could potentially be a better choice. Given that the RNN surpassed the CNN in terms of performance after 4 epochs, this could perhaps make the tradeoff in computational time worth it (the RNN processing time for 4 epochs was about 27 minutes).

6 References

[1] *Phishing SS Bug - CISA*, www.cisa.gov/sites/default/files/2023-02/phishing-infographic-508c.pdf. Accessed 8 Dec. 2023.

[2] Y. Huang, Q. Yang, J. Qin and W. Wen, "Phishing URL Detection via CNN and Attention-Based Hierarchical RNN," 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), Rotorua, New Zealand, 2019, pp. 112-119, doi: 10.1109/TrustCom/BigDataSE.2019.00024.

[3] Alshingiti, Z.; Alaqel, R.; Al-Muhtadi, J.; Haq, Q.E.U.; Saleem, K.; Faheem, M.H. A Deep Learning-Based Phishing Detection System Using CNN, LSTM, and LSTM-CNN. *Electronics* **2023**, *12*, 232. https://doi.org/10.3390/electronics12010232

[4] Ariyadasa, Subhash, Subha Fernando, and Shantha Fernando. "Detecting phishing attacks using a combined model of LSTM and CNN." *Int. J. Adv. Appl. Sci* 7.7 (2020): 56-67.

[5] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825–2830. Retrieved from http://jmlr.org/papers/v12/pedregosa11a.html

[6] Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras