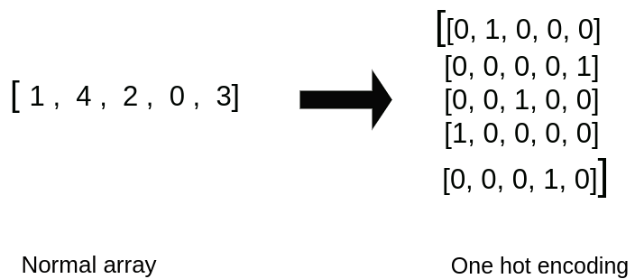
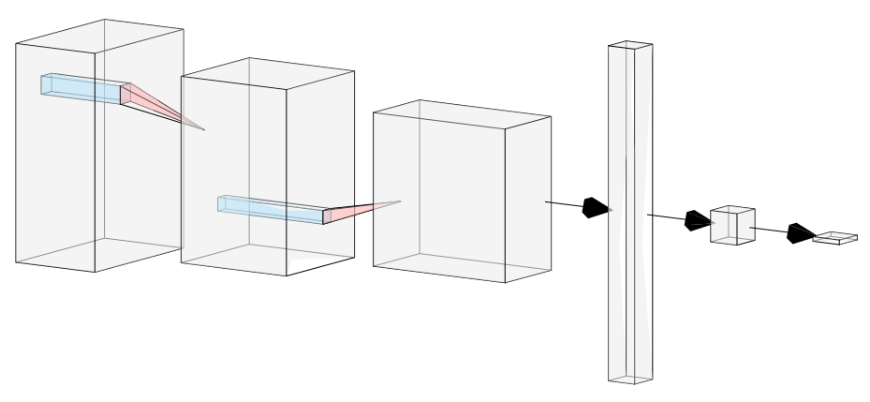
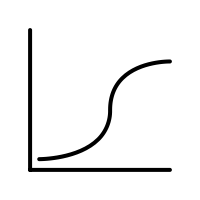
CMPE 452 Final Report

Phishing URL Detection using a Convolutional Neural Network

John Turnbull – 20235355

Alex Fullerton – 20168873



# Motivation

This project is driven by the increasing prevalence of phishing attacks involving dubious Uniform Resource Locators (URLs). Individuals are consistently falling victim to these deceptive schemes, resulting in financial losses, jeopardized personal information, unauthorized access to secure accounts, and breaches of operating systems due to viruses. In 2022, reported phishing crimes alone led to nearly $52 million in losses [3]. Our aim is to counteract this trend. Our project is dedicated to developing improved methods for identifying malicious links and equipping everyone with the necessary tools to ensure online safety. We aspire to deliver this service through our neural network, which provides a precise method of detection against these harmful links.

# Problem Description

Convolutional neural networks (CNNs) are primarily known for their effectiveness in tasks involving feature extraction in images. The design goal for this project is to build a neural network motivated by the structure of a CNN to detect harmful phishing URLs. Many approaches to phishing detection involve preprocessing to catalogue URL features, retrieving additional information about the email or webpage, or consulting external blacklist sources. This neural network will take only a string of characters (the URL) as its input making it much more space efficient, reliable, and easy to implement than other approaches.

# Contribution

The project's contributions were shared between group members with substantial progress being achieved jointly during weekly working sessions. It's worth noting that projects of this nature are typically designed for groups of four, putting us at a disadvantage and forcing our group to shoulder a larger workload to completion. The contributions of each group member to the project are catalogued in the table below.

|  |  |
| --- | --- |
| Groupmate | Contribution |
| John Turnbull | Data acquisitionData preprocessingEncoder implementation |
| Alex Fullerton | Data augmentationModel creationTraining & Validation  * Testing |

# Related Work

Our project drew inspiration from the report titled "Accurate and Fast URL Phishing Detector: A Convolutional Neural Network Approach,"[1] authored by a group of specialists in machine learning. This comprehensive report supplied all the necessary knowledge to implement a neural network model capable of effectively classifying phishing URLs. The study compared two models, namely LSTM (long short-term memory) and CNN (convolutional neural network). They elected to present the data in the form of a character level one-hot vector spanning 70 characters. The CNN model consisted of 9 layers in total from a 256 x 70 input to a 2 x 1 output. This included use of fully connected layers, convolutions, and max pooling. The dataset contained approximately 21200 records balanced between harmful and safe, and it was divided into 80% training and 20% testing. The CNN model was trained and tested over 15 epochs, taking 3.5 minutes to train and it yielded an impressive accuracy of 99.98%. The authors concluded that CNN outperformed LSTM due to its ease of training and faster convergence of the training curve. Their CNN model achieved an accuracy of 99.98% compared to their LSTM model which scored an accuracy of 98.5%.

Shifting focus to the second report, "Classifying Phishing URLs using Recurrent Neural Networks,"[2] authored by a similar group of researchers tackling the same issue. However, this report introduced a comparison between LSTM and another model, Random Forest. These experiments were conducted with a dataset of 1 million normal URLs and 1 million phishing URLs. The RF model achieved an accuracy of 93.5% compared to the LSTM model which achieved an accuracy of 98.7%. It's crucial to note the publication date difference, with this paper being from 2017 compared to the one mentioned before being a 2020 paper. While the conclusions favored the LSTM model, given the known superiority of CNN from the primary report, this paper contributed minimal abstract knowledge that we could apply during implementation.

# Datasets Used

Initially, our dataset included various additional attributes associated with the URLs. However, we opted to streamline the dataset to include only the label and the link. This decision was motivated by the desire to facilitate real-time model implementation, as relying solely on URLs as input simplifies the process. The initial dataset featured approximately 11,000 unique links, evenly distributed between phishing and non-phishing links. However, due to the dataset's limited size, the model's accuracy was constrained.

Eventually, we decided to transition to a larger dataset comprising of around 190,000 unique URLs. With this expanded dataset, we observed a substantial increase in model accuracy. We sourced the new dataset from Kaggle, a platform where it produced a perfect usability score based on factors such as completeness, credibility, and compatibility. The dataset is evenly split, containing 95,000 phishing links and 95,000 regular links. The label is represented in bitwise form, with 0 indicating a non-phishing link and 1 indicating a phishing link.

A screenshot of a computer

Description automatically generated

Figure 1: Larger dataset that we ended up using for our final iteration of the project.

# Data Preprocessing

The chosen model architecture is motivated by a convolutional neural network. These types of networks use a sliding window, called a filter, to extract features from data. They are known for their effectiveness specifically on feature extraction in images.

Since we are working with strings instead of images, we need to transform the data into a structure that makes sense for implementation with a CNN. The solution is to use one-hot encoding with a dictionary of all the characters which may be found in a URL to define the value associated with each character. The dictionary used for the project is shown below:

**Dictionary: abcdefghijklmnopqrstuvwxyz\_0123456789-;.!?:/\\|#$%^&~’+=<>(),"’|^**

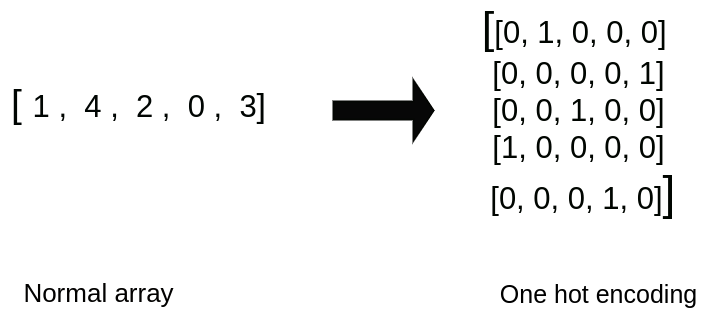
**0 … 64**

The dictionary, consisting of 64 characters, assigns a character to each index in the one hot vector. Each character in the URL is one hot encoded resulting in a 2D tensor like what is shown in Figure 2.

Figure 2 An illustration depicting the transformed tensor of one hot vectors generated from a URL input.

www.faqs.org/faqs/comp...

C



These resultant tensors can be treated like an image from which features may be extracted to classify harmful URLs.

Before this data can be implemented, the input size needs to be standardized across the dataset, which is a challenge since the URLs are of widely varying lengths. The solution implemented was to set a maximum length of 256, cutting off any characters afterward, and to zero pad any URLs shorter than 256. URLs longer than 256 characters accounted for 0.72% of our dataset so very little information was lost by clipping longer URLs. Additionally, appending vectors of zeros to the end of shorter URLs will not affect any of the existing information which is vital to classify the data. By implementing this solution, all the tensors in the dataset now had the shape 256 x 64.

# Experimental Setup

The final model consists of 3 convolutional layers with a ReLU activation function and max pooling applied after each layer. ReLU is a common activation function given by:

Max pooling is another machine learning technique where only the maximum value in a given window is selected as the output. This process, which accounts for the decrease in spatial dimension between convolutional layers, is illustrated in Figure 3.

Figure 3 A graphic which explains the process of max pooling [4].

The output of the final convolutional layer is flattened into a single 1 x 1664 vector which is fed into a fully connected layer. The model terminates at a single output node. A sigmoid function is applied to the output node to compress the data between 0 and 1. This way, the model acts as a binary classifier where URLs with outputs greater than 0.5 are classified as phishing URLs, while outputs less than 0.5 are considered legitimate. The model was implemented using PyTorch. A diagram of the final model architecture is depicted in Figure 4 and a summary of the model layers is included in Appendix A.

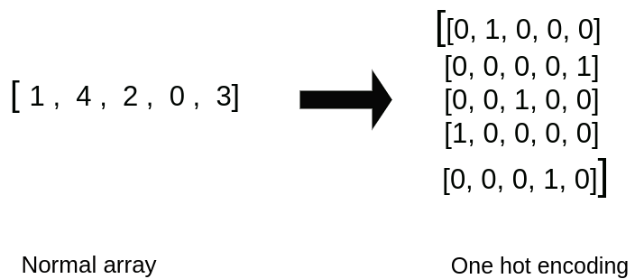
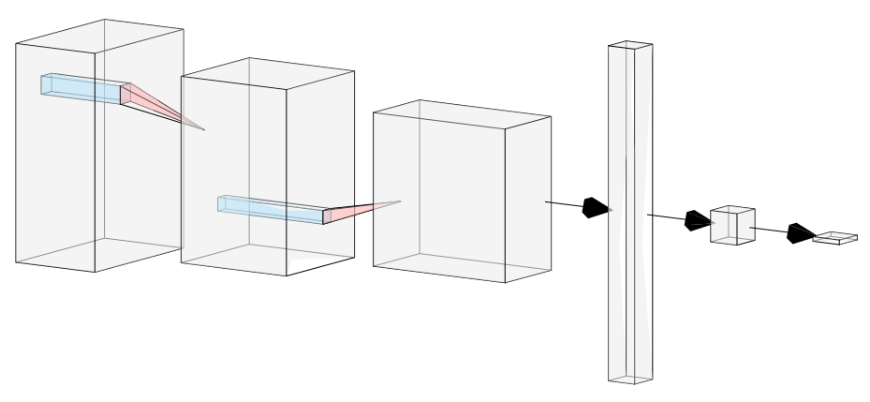
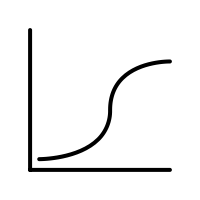


Figure 4 A schematic of the final model architecture.

# Training/Testing/Validation

A graph of training and validation loss

Description automatically generated The model was run for 60 epochs with a batch size of 128. The SGD and Adam optimizers were both experimented with, and better results were obtained with Adam with a learning rate of 1.5e-3. This was paired with the exponential learning rate scheduler with a gamma of 0.9, meaning the learning rate is decreased by a factor of 0.9 after each epoch to avoid getting stuck in a local minima. Binary cross entropy loss was implemented as the loss function for this task as it is effective for binary classification problems such as this. The loss curves associated with the top training attempt are displayed in Figure 5. After 60 epochs the loss curves are very flat and know consistent improvement was observed. The training, validation and testing datasets were randomly split from a single dataset with a ratio of 0.9 : 0.05 : 0.05. The training dataset is kept as large as possible to try to maximize the effectiveness of training.

Figure 5 Training and validation loss curves for the top training attempt.

The trained model weights were then frozen and run on the test dataset with a threshold of 0.5 to classify the URLs as being either phishing or legitimate. The accuracy was calculated by dividing the number of correct predictions by total number of URLs, and was found to be 94.69%

# Results and Discussion

The results of the model were very positive. Through iterative training, hyperparameter adjustments, tweaks in the model architecture, and improvements in data processing, the final trained model achieved a classification accuracy of 94.69% and a test loss of 0.149 over 60 epochs. This result represents a marked improvement over the 50% probability of a random guess. The test loss is comparable to the training and validation losses indicating that the model was not overfitting too much to the training dataset.

|  |  |
| --- | --- |
| Accuracy | 94.69% |
| Test Loss | 0.149 |

Table 1 Testing results for the final model.

Increasing the size of the training dataset had a massive effect on the results during testing. The model was originally trained on a dataset containing ~11000 unique links and the resulting accuracy during testing was around 82%. Increasing the dataset size to ~190000 links resulted in a 12% increase in classification accuracy. This diverse training set allowed the model to generalize well to unseen data, contributing significantly to its high accuracy on the test set. The confusion matrix for URL classification on the test dataset is displayed below.

The model demonstrates the ability to correctly classify both phishing and legitimate URLs. The model classifies phishing URLs correctly 96.06% of the time and legitimate URLs 93.4% of the time. This skew toward phishing classification is preferable for a model with an application in security. The metrics associated with the confusion matrix are captured in Table 2.

Table 2 Confusion matrix metrics from model testing.

|  |  |
| --- | --- |
| Error Rate | 5.31% |
| Accuracy | 94.69% |
| Recall | 96.06% |
| Specificity | 93.4% |
| Precision | 93.15% |

The approach of using a CNN architecture proved to be effective as the model was able to recognize patterns inherent in the phishing URL structures which were discernible from those of legitimate URLs.

# Conclusion and Future Work

Our group is very satisfied with the final performance of our system, especially given our small group size. The goal of the project to build a model to classify phishing fraud attempts using only the URLs was accomplished by the implementation of a CNN as proposed in the problem statement. The resultant model has the capability to serve as a valuable tool to users to evaluate potential phishing schemes.

Though the model did perform well, it did not match the accuracy of the model implemented in the paper from which we drew inspiration, listed at 99.79% [1]. Some next steps we could take which we think may improve performance includes implementing an embedding layer to decrease the size of the input tensor before feeding it through the convolutional layers. The paper was unspecific about how they did this without losing sequential information inherent in the URL necessary to extract features. We struggled to emulate this and ultimately scrapped it in our final model. Training on a larger dataset would also increase the capability of the model to generalize which will improve performance between training and testing datasets. Using other machine learning techniques such as dropout and regularization to reduce overfitting may also improve performance.

# References

[1] [Wei, W., Ke, Q., Nowak, J., Korytkowski, M., Scherer, R., & Woźniak, M. (2020). *Accurate and fast URL phishing detector: A Convolutional Neural Network Approach. Computer Networks*, *178*, 107275. <https://doi.org/10.1016/j.comnet.2020.107275>]

[2] [Bahnsen, A. C., Bohorquez, E., Villegas, S., Vargas, J., & Gonzalez, F. (2017, April 27). *Classifying phishing urls using recurrent neural networks*. ieeexplore. <https://ieeexplore.ieee.org/document/7945048>]

[3] [Gendre. (n.d.). Spear Phishing vs Phishing: What’s a Greater Threat for Your Business. Retrieved December 7, 2023, from <https://www.vadesecure.com/en/blog/spear-phishing-vs-phishing#:~:text=Cost%3A%20Spear%20phishing%20and%20phishing,billion%20reported%20by%20BEC%20victims>]

[4] “Papers with Code - Max Pooling Explained,” paperswithcode.com. https://paperswithcode.com/method/max-pooling

‌

# Appendix A – Model Architecture

A computer screen shot of white text

Description automatically generatedConvolutional layer notation is: (input depth, output depth, kernel size, stride)