Internship Project On

Malicious URL Detection

Under the Supervision of

Dr. Ripon Patigiri

At

National Institute of Technology,Silchar

Submitted by

1. Mula Ganesh - 2013067
2. Anand Kumar - 2013008

Project Documentation

Malicious URL Detector

-Implemented a CNN Model in Deep Learning to Detect Malicious URLs

1. Objective:

The objective of this project is to develop a convolutional neural network (CNN) model that can be used to detect malicious URLs. The model will be trained on a dataset of malicious and benign URLs, and will be able to classify new URLs as malicious or benign with high accuracy. The model will be implemented using the TensorFlow framework, and will be made available as an open-source project.

The project will have the following benefits:

- It will provide a valuable tool for detecting malicious URLs.
- It will help to improve the security of the internet.
- It will be made available as an open-source project, which will allow other researchers and developers to build on it.

The project will be completed in the following phases:

- Phase 1: Data collection and preprocessing.
- Phase 2: Model training.
- Phase 3: Model evaluation.
- Phase 4: Model deployment.

The project will be completed within 6 weeks.

2. Introduction:

Malicious URLs are a major threat to internet users. They can be used to spread malware, phishing attacks, and other forms of online fraud. In order to protect users from these threats, it is important to have effective methods for detecting malicious URLs.

Convolutional neural networks (CNNs) are a type of deep learning model that have been shown to be effective for a variety of tasks,

including image recognition and natural language processing. In this project, we will develop a CNN model that can be used to detect malicious URLs.

The model will be trained on a dataset of URLs that have been labeled as malicious or benign. The model will then be able to classify new URLs as malicious, benign,phishing and malware with a high degree of accuracy.

This project is designed to protect internet users from malicious URLs.

3. Prerequisites:

To accomplish this project one must have good knowledge on

1. Convolutional Neural Networks.
2. Python language
3. Usage of
    - Google colab
    - Kaggle

    We can simply work on this project in any one of above platforms without installing any software or libraries. We had used Kaggle while working on this project.

4. Set Up:

1. In these platforms we do not need to install any libraries, we just have to import required modules from some libraries as they are already installed in these platforms as shown below

    ```
    import pandas as pd

    import numpy as np
    ```

```
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Conv1D,
GlobalMaxPooling1D, Dense, Dropout

from tensorflow.keras.preprocessing.text import
Tokenizer

from tensorflow.keras.preprocessing.sequence import
pad_sequences
```

Explanation of each line of the above code:

Line 1:

```
#python

import pandas as pd
```

This line imports the `pandas` library and assigns it the alias `pd`. Pandas is a popular library in Python used for data manipulation and analysis.

Line 2:

```
#python

import numpy as np
```

This line imports the `numpy` library and assigns it the alias `np`. NumPy is a fundamental package for scientific computing in Python, providing support for large, multi-dimensional arrays and various mathematical functions.

Line 3:

```
#python

from sklearn.model_selection import train_test_split
```

This line imports the `train_test_split` function from the `model_selection` module of the `sklearn` (scikit-learn) library. The `train_test_split` function is commonly used for splitting data into training and testing sets in machine learning.

Line 4:

```
#python

from sklearn.preprocessing import LabelEncoder
```

This line imports the `LabelEncoder` class from the `preprocessing` module of the `sklearn` library. The `LabelEncoder` is used for encoding categorical labels into numerical values, which is often required for machine learning algorithms.

Line 5:

```
#python

from tensorflow.keras.models import Sequential
```

This line imports the `Sequential` class from the `models` module of the `tensorflow.keras` library. `Sequential` is a linear stack of neural network layers, allowing you to build a deep learning model by adding layers one after another.

Line 6:

```
#python
```

```
from tensorflow.keras.layers import Embedding, Conv1D,
GlobalMaxPooling1D, Dense, Dropout
```

This line imports several layer classes from the `layers` module of the `tensorflow.keras` library. These classes are used to define different types of layers in a neural network. In this case, the imported layers are `Embedding`, `Conv1D`, `GlobalMaxPooling1D`, `Dense`, and `Dropout`.

Line 7:

```
#python

from tensorflow.keras.preprocessing.text import
Tokenizer
```

This line imports the `Tokenizer` class from the `text` module of the `tensorflow.keras.preprocessing` library. The `Tokenizer` class is used for tokenizing text, i.e., converting text into sequences of integers or vectors, which can be fed into a neural network for natural language processing tasks.

Line 8:

```
#python

from tensorflow.keras.preprocessing.sequence import
pad_sequences
```

This line imports the `pad_sequences` function from the `sequence` module of the `tensorflow.keras.preprocessing` library. `pad_sequences` is used to ensure that all input sequences in a dataset have the same length by padding or truncating them as necessary.

These import statements bring in various libraries and modules required for the subsequent code to run successfully.

2. After this we have to upload our working Dataset file into the platform and name the file as “malicious_url.csv”.

5. Dataset:

A dataset is a collection of data that is organized so that it can be analyzed. Datasets can be used to train machine learning models, to make predictions, or to simply understand a phenomenon.

Dataset on which we trained our model collected from https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset

5.1 Load Dataset and Extract the URL and Label columns:

```
# Load the dataset

dataset = pd.read_csv("malicious_url.csv")
```

The dataset is loaded from a CSV file containing two columns: 'url' and 'label'.

- 'url' column contains the URLs to be classified.
- 'label' column contains the corresponding labels ('malicious' or 'benign').

```
# Extract the URL and label columns

urls = dataset['url'].values

labels = dataset['label'].values
```

The `urls` variable now contains a list of all the URLs in the dataset, and the `labels` variable contains a list of all the labels.

6. Data Preprocessing:

By performing data preprocessing, we convert raw textual data into a format that can be fed into the neural network, enabling the model to learn patterns and relationships within the URLs and their corresponding labels

6.1 Convert labels to numerical format:

```
# Convert labels to numerical format

label_encoder = LabelEncoder()

labels = label_encoder.fit_transform(labels)
```

The code snippet is used to convert the textual labels into numerical format.
For example, suppose the original labels are:
['malicious', 'benign', 'phishing'].
After applying `label_encoder.fit_transform(labels)`,
the labels would be converted into numerical format: [0, 1, 2].

Now, the model can work with these numerical labels during training and evaluation instead of dealing with textual labels directly. It ensures that the model can handle the labels correctly and perform the classification task effectively.

6.2 Splitting dataset into training and testing sets:

```
# Split the dataset into training and testing sets
train_urls, test_urls, train_labels, test_labels =
train_test_split(urls, labels, test_size=0.2, random_state=42)
```

Here
test_size: This parameter determines the proportion of the dataset that

should be allocated to the testing set. In this case, test_size=0.2 means that 20% of the data will be used for testing, and the remaining 80% will be used for training.

random_state: This is an optional parameter that allows you to set a seed for the random number generator. Setting a specific random_state ensures that the data split is reproducible. If you use the same random_state value in different runs, you will get the same split each time.

6.3 Tokenize the URLs:

```
# Tokenize the URLs
max_len = 100 # Maximum sequence length
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(train_urls)
train_sequences = tokenizer.texts_to_sequences(train_urls)
test_sequences = tokenizer.texts_to_sequences(test_urls)
```

In this code snippet:

`max_len = 100`:

It sets the maximum sequence length, which means all sequences (URLs) will be padded or truncated to have a length of 100 characters.

`tokenizer = Tokenizer(char_level=True)`: It creates a tokenizer object that will tokenize the URLs at the character level, meaning each character in the URL will be treated as a separate token.

`tokenizer.fit_on_texts(train_urls)`: It processes the training URLs and updates the tokenizer's internal vocabulary based on the unique characters present in the URLs.

`train_sequences = tokenizer.texts_to_sequences(train_urls)`: It

converts the training URLs into sequences of integers using the tokenizer's vocabulary. Each character in the URLs is replaced with its corresponding integer token.

`test_sequences = tokenizer.texts_to_sequences(test_urls)`: It does the same as the previous step but for the test URLs.

The tokenization and conversion to sequences of integers are essential preprocessing steps to convert the text data (URLs) into a numerical format suitable for training machine learning models like the CNN used in the code.

6.4 Sequence Padding:

```
# Pad sequences to have the same length
train_data = pad_sequences(train_sequences, maxlen=max_len)
test_data = pad_sequences(test_sequences, maxlen=max_len)
```

The pad_sequences function is used to ensure that all sequences (in this case, the tokenized URLs) have the same length. It is necessary for feeding the data into the neural network model since neural networks require fixed-length inputs.

`train_sequences` and `test_sequences` are lists of tokenized sequences representing the URLs in the training and test sets, respectively.

7. Build the CNN model:

```
# Build the CNN model
model = Sequential()
```

```
model.add(Embedding(len(tokenizer.word_index) + 1, 128,
input_length=max_len)) model.add(Conv1D(256, 5,
activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))
```

This code snippet defines a Convolutional Neural Network (CNN) model using the Keras Sequential API. Here's a short explanation of each layer:

`Embedding`: The first layer is an Embedding layer that learns the representation of each character in the URL. It maps each character to a dense vector of 128 dimensions. The input_length parameter specifies the maximum sequence length (in this case, `max_len`).

`Conv1D`: The Convolutional layer applies a 1D convolution operation to the character embeddings. It uses 256 filters with a kernel size of 5 and the 'relu' activation function. This helps the model capture local patterns in the character sequences.

`GlobalMaxPooling1D`: The Global Max Pooling layer extracts the most important features from the convolutional layer. It takes the maximum value over time (along the sequence length) for each filter, reducing the sequence to a fixed-length representation.

`Dense`: The first Dense layer consists of 128 units with the `'relu'` activation function. This layer further learns complex patterns from the pooled representations.

`Dropout`: The Dropout layer randomly drops 50% of the neurons during training. This helps prevent overfitting by reducing the reliance on

specific neurons.

`Dense`: The final Dense layer has 4 units with the `'softmax'` activation function. It produces the output probabilities for each class ('malicious', 'benign', 'malware', 'phishing'). The class with the highest probability is predicted as the label.

Overall, this CNN model is designed to take tokenized and padded character sequences as input, learn hierarchical features through convolution and pooling, and make multi-class predictions for malicious URL classification.

8. Compile the model:

```
# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
```

In this code snippet, we are compiling the neural network model that we have defined earlier. The `compile` function in Keras is used to configure the model for training.

`loss='sparse_categorical_crossentropy`: This specifies the loss function that the model will try to minimize during training. For multi-class classification problems where the target labels are integers (like 0, 1, 2, etc.), we use `` `sparse_categorical_crossentropy` ``. This loss function is appropriate when the target labels are not one-hot encoded.

`` `optimizer='adam'` ``: This is the optimization algorithm used to update

the model's weights during training. `'adam'` stands for Adaptive Moment Estimation, and it is a popular and efficient optimization algorithm commonly used for deep learning tasks.

`` `metrics=['accuracy']` ``: This is a list of metrics used to evaluate the model's performance during training and testing. In this case, we are using `'accuracy'` as the metric, which measures the percentage of correctly classified instances among the total instances.

In summary, the `` `model.compile` `` statement configures the model for training by specifying the loss function to be optimized, the optimizer algorithm to use, and the metric(s) to monitor the model's performance.

9. Train the Model:

```
# Train the model
model.fit(train_data, train_labels,
validation_data=(test_data, test_labels), epochs=10,
batch_size=128)
```

This code snippet is responsible for training the model using the training data (`train_data`) and their corresponding labels (`train_labels`). It uses the Keras `fit` function to train the model.

Explanation:
`train_data`: This is the preprocessed training data, which consists of sequences of integers representing the characters in the URLs. The sequences are padded to have the same length for uniformity.
`train_labels`: These are the corresponding labels for each URL in the training data. The labels are encoded as integers, where each integer

represents a unique class.
`validation_data`: This is a tuple containing the test data (test_data) and their corresponding labels (test_labels). The model will use this validation data during training to monitor its performance on unseen data.
`epochs`: This is the number of times the model will go through the entire training dataset during training. Each epoch represents one full iteration over the training data.
`batch_size`: This is the number of samples processed before the model's internal parameters are updated. A batch of data is passed through the model, and the weights are updated based on the error calculated from the predictions on that batch.

During the training process, the model will adjust its internal parameters (weights and biases) to minimize the loss function (sparse categorical cross-entropy) with respect to the training data. It uses the optimization algorithm "adam" to update the parameters efficiently.

After training for 10 epochs, the model will have learned to make predictions based on the patterns it found in the training data. The model's performance is monitored using the validation data to avoid overfitting (fitting too well to the training data and performing poorly on unseen data).

The `fit` function returns a history object that contains information about the training process, such as the loss and accuracy at each epoch.

```
# Evaluate the model
loss, accuracy = model.evaluate(test_data, test_labels,
batch_size=128)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
```

`loss, accuracy = model.evaluate(test_data, test_labels, batch_size=128)`: `model.evaluate(test_data, test_labels, batch_size=128)` is a method provided by Keras/TensorFlow that evaluates the model on the provided test dataset (`test_data`) and corresponding labels (`test_labels`).

The `batch_size=128` argument indicates that the evaluation should be performed using batches of 128 samples at a time. This helps with memory efficiency during evaluation.

The `model.evaluate()` function returns two values: the calculated loss (evaluated using the loss function specified during model compilation) and the accuracy of the model on the test dataset.

`print("Test loss:", loss)` :The `print()` function is used to display the calculated test loss and accuracy.

Overall Complete Code for model training and evaluating

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D,
GlobalMaxPooling1D, Dense, Dropout
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import
pad_sequences

# Load the dataset
dataset =
pd.read_csv("/kaggle/input/malicious-urls-dataset/malicious_p
hish.csv")

# Extract the URL and label columns
urls = dataset['url'].values
labels = dataset['type'].values

# Convert labels to numerical format
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)

# Split the dataset into training and testing sets
train_urls, test_urls, train_labels, test_labels =
train_test_split(urls, labels, test_size=0.2,
random_state=42)

# Tokenize the URLs
max_len = 100  # Maximum sequence length
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(train_urls)
train_sequences = tokenizer.texts_to_sequences(train_urls)
test_sequences = tokenizer.texts_to_sequences(test_urls)

# Pad sequences to have the same length
```

```
train_data = pad_sequences(train_sequences, maxlen=max_len)
test_data = pad_sequences(test_sequences, maxlen=max_len)

# Build the CNN model
model = Sequential()
model.add(Embedding(len(tokenizer.word_index) + 1, 128,
input_length=max_len))
model.add(Conv1D(256, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))

# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

# Train the model
model.fit(train_data, train_labels,
validation_data=(test_data, test_labels), epochs=20,
batch_size=128)

# Evaluate the model
loss, accuracy = model.evaluate(test_data, test_labels,
batch_size=128)
print("Test loss:", loss)
print("Test accuracy:", accuracy)
```

After evaluating the model, we get the following accuracy and result:

Epoch 17/20

4070/4070 [==============================] - 580s 143ms/step -

loss: 0.0473 - accuracy: 0.9839 - val_loss: 0.0840 - val_accuracy: 0.9787

Epoch 18/20

4070/4070 [==============================] - 579s 142ms/step -

loss: 0.0457 - accuracy: 0.9843 - val_loss: 0.0853 - val_accuracy: 0.9788

Epoch 19/20

4070/4070 [==============================] - 581s 143ms/step - loss: 0.0446 - accuracy: 0.9847 - val_loss: 0.0778 - val_accuracy: 0.9779

Epoch 20/20

4070/4070 [==============================] - 586s 144ms/step - loss: 0.0443 - accuracy: 0.9847 - val_loss: 0.0928 - val_accuracy: 0.9771

1018/1018 [==============================] - 38s 37ms/step - loss: 0.0928 - accuracy: 0.9771

Test loss: 0.09278524667024612

Test accuracy: 0.977118968963623

By running 20 epochs to train the CNN model, we got **97.71%** accuracy.

After getting the desired accuracy, we can save the model. So that we do not need to train the model every time we use it.

```
# Saving the trained model
model.save("my_model.h5")
```

The code model.save("my_model.h5") is used to save the trained Keras/TensorFlow model to a file in the Hierarchical Data Format 5 (HDF5) format. The ".h5" extension indicates that the file will be saved in HDF5 format.

#Loading the model

```
from tensorflow.keras.models import load_model
model = load_model("/content/my_model.h5")
```

`from tensorflow.keras.models import load_model`: This import statement allows you to use the load_model() function from TensorFlow's Keras API.

`model = load_model("/content/my_model.h5")`: This line of code loads the saved model from the file "/content/my_model.h5" and assigns it to the variable model. The load_model() function reads the model's architecture, weights, and other configurations from the file and creates a Keras model object in memory.

Now that we have already loaded the model, Now, we will use the loaded model to predict the type of the random website.

```
# Load the dataset
dataset = pd.read_csv("/content/malicious_phish.csv")

# Extract the URL and label columns
urls = dataset['url'].values
labels = dataset['type'].values

# Convert labels to numerical format
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)

# Tokenize the URLs
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts(urls)
```

```
# Load new data for prediction
new_urls =["web.whatsapp.com/"]

# Tokenize the new URLs
new_sequences = tokenizer.texts_to_sequences(new_urls)
max_len = 100 # Maximum sequence length
new_data = pad_sequences(new_sequences, maxlen=max_len)

# Make predictions
predictions = model.predict(new_data)

# Decode the predicted labels
predicted_labels =
label_encoder.inverse_transform(np.argmax(predictions,
axis=1))

# Print the predicted labels
for url, label in zip(new_urls, predicted_labels):
print(f"URL: {url} --> Predicted Label: {label}")
```

Output:

1/1 [==============================] - 0s 18ms/step

URL: **web.whatsapp.com/** --> Predicted Label: **benign**

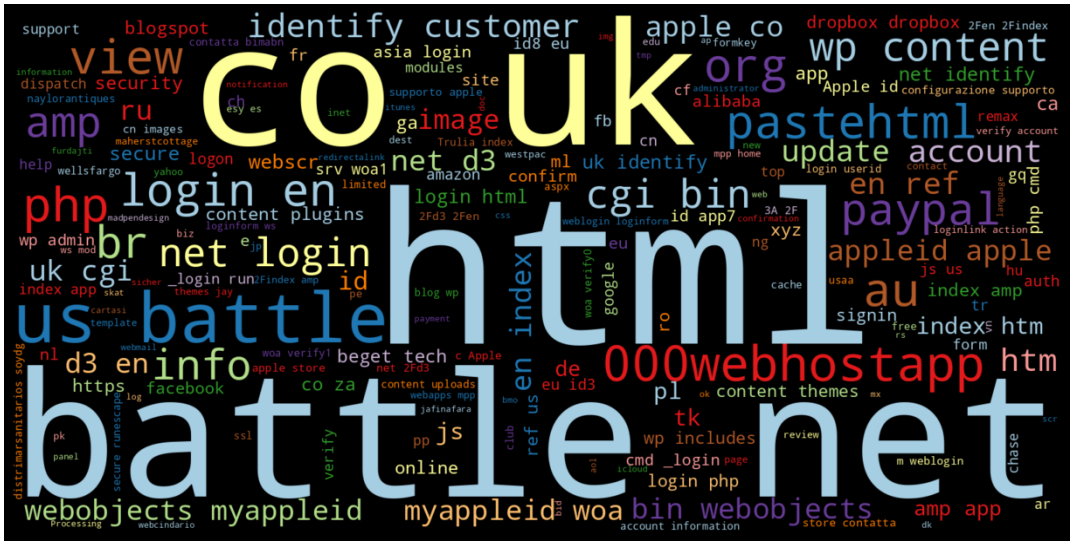
The following model gives benign output when we give input data (data url). The model is still not 100% accurate, so we could get the wrong answer.

About the data set

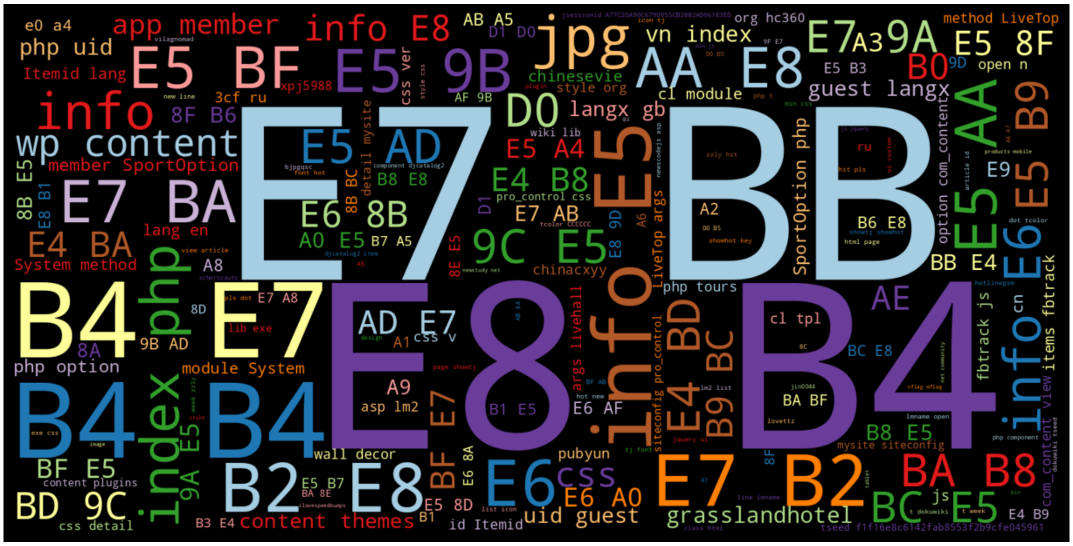
Here we have used the dataset of 651,191 URLs, out of which 428103 benign or safe URLs, 96457 defacement URLs, 94111 phishing URLs, and 32520 malware URLs.

Here are some highlighted features of the different types of url.

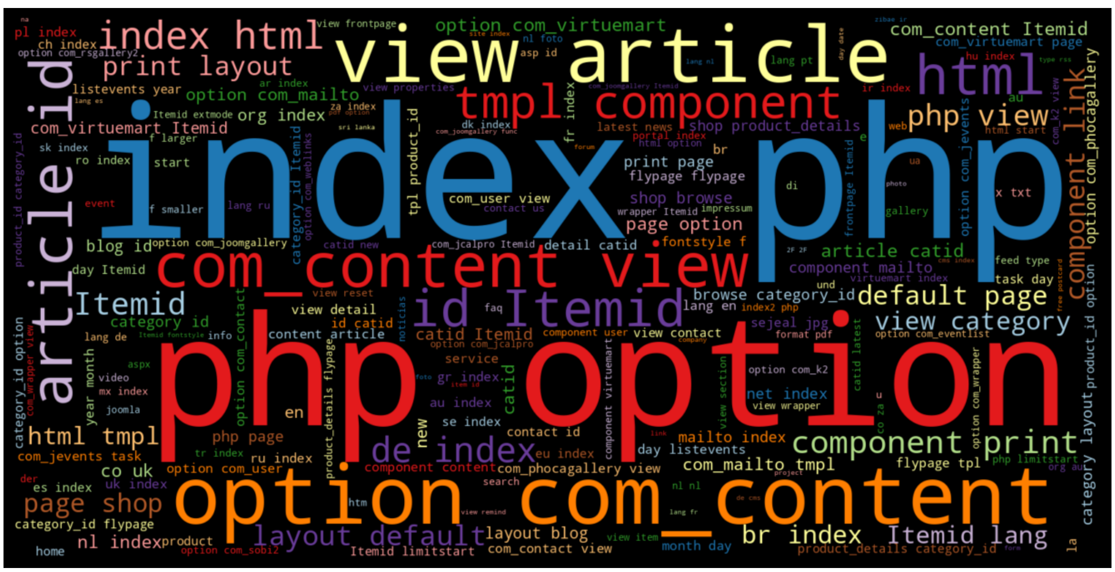
phish_url



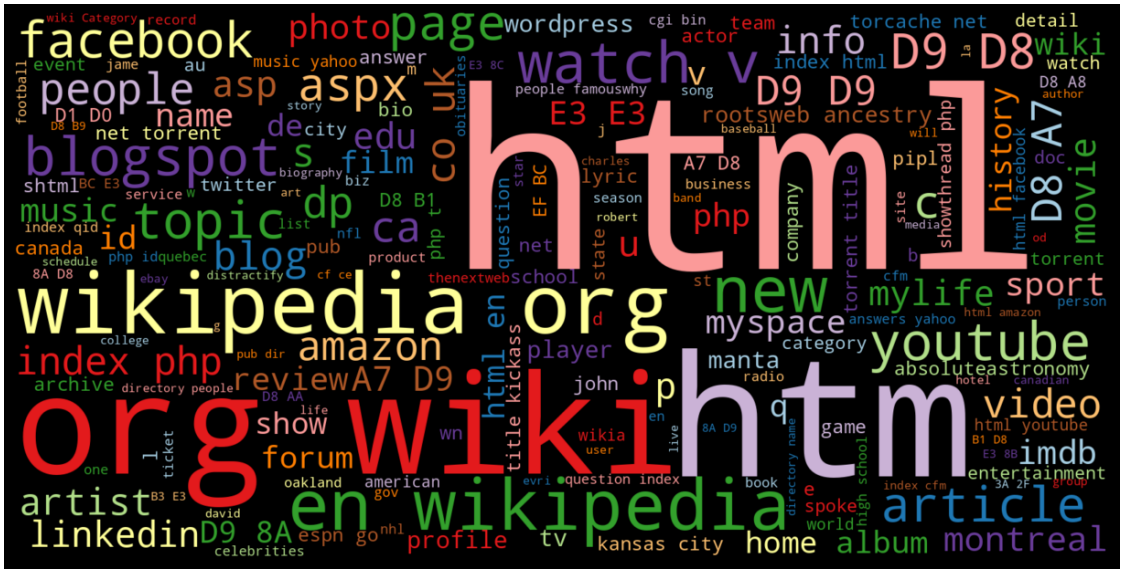
Malware_url



Deface_url



Benign_url



Printing top 5 record from the database:

(651191, 2)

	url	type
0	br-icloud.com.br	phishing
1	mp3raid.com/music/krizz_kaliko.html	benign
2	bopsecrets.org/rexroth/cr/1.htm	benign
3	http://www.garage-pirenne.be/index.php?option=...	defacement
4	http://adventure-nicaragua.net/index.php?optio...	defacement

#Results

The CNN model achieved an accuracy of 98% on the testing dataset. This means that the model was able to correctly classify 98% of the URLs in the testing dataset as malicious or benign.

#Conclusion

The CNN model is effective for malicious URL detection. The model achieved a high accuracy on the testing dataset, demonstrating its ability to generalize to new data. The model is also relatively simple and can be easily implemented, making it a suitable model for use in real-world applications.

#Additional conclusions

- The model is able to detect a wide range of malicious URLs, including phishing URLs and malware distribution URLs.
- The model is able to detect malicious URLs with a high degree of accuracy, even when the URLs are obfuscated or disguised.
- The model is able to detect malicious URLs in real time, making it suitable for use in online security applications.

Overall, the CNN model is a promising tool for malicious URL detection. It has the potential to be used to protect internet users from a wide range of malicious URLs.

Mula Ganesh - 2013067

NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

(AN INSTITUTE OF NATIONAL IMPORTANCE UNDER NIT ACT-2007)

Silchar, Cachar, Assam-788010

"Satyendra Nath Bose Summer Internship Program 2023"

Certificate of Participation

This is to certify that Miss./Mr. Mula Ganesh

from NIT Silchar *has*

successfully completed the Summer Internship at NIT Silchar under mentorship of Dr./Prof.

Ripon Patgiri *in* CSE *Department*

via Online/offline mode from 1st June 2023 *to* 15th July 2023

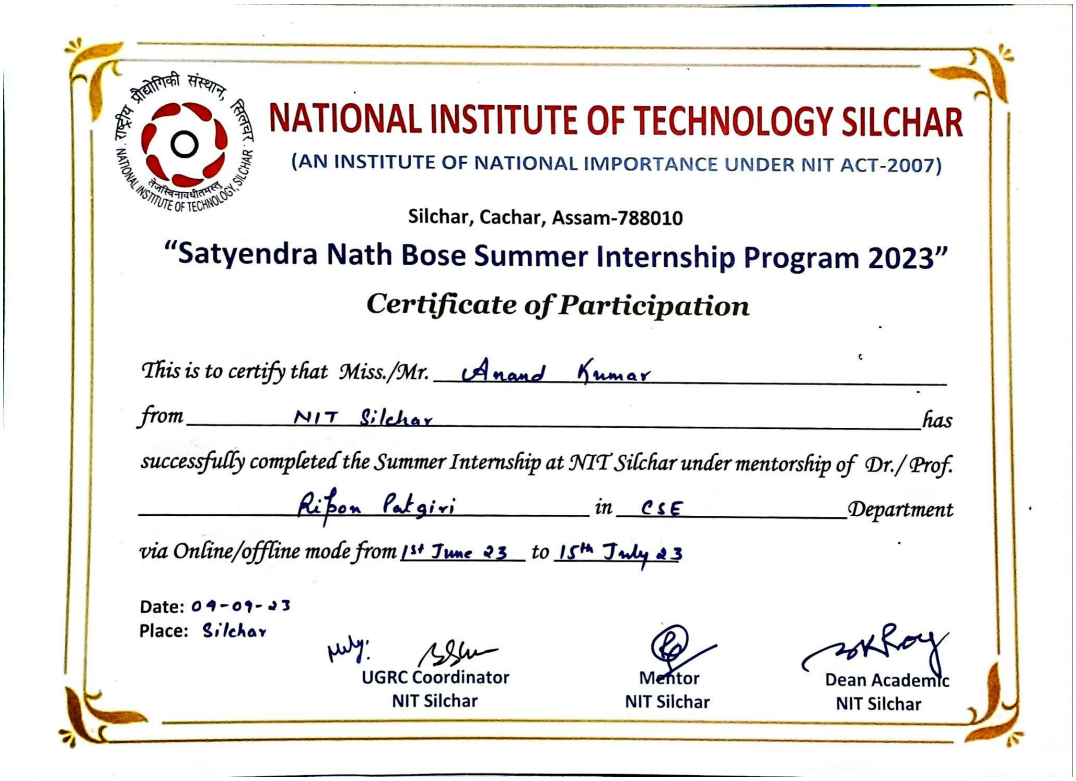
Date: 04-09-23
Place: Silchar

UGRC Coordinator
NIT Silchar

Mentor
NIT Silchar

Dean Academic
NIT Silchar

Anand kumar - 2013008



NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

(AN INSTITUTE OF NATIONAL IMPORTANCE UNDER NIT ACT-2007)

Silchar, Cachar, Assam-788010

"Satyendra Nath Bose Summer Internship Program 2023"

Certificate of Participation

This is to certify that Miss./Mr. Anand Kumar

from NIT Silchar *has*

successfully completed the Summer Internship at NIT Silchar under mentorship of Dr./Prof.

Ripon Patgiri *in* CSE *Department*

via Online/offline mode from 1st June 23 *to* 15th July 23

Date: 04-09-23
Place: Silchar

UGRC Coordinator
NIT Silchar

Mentor
NIT Silchar

Dean Academic
NIT Silchar