B.E. CSE VI – P BATCH

CS6611 – CREATIVE AND INNOVATIVE PROJECT

Malicious URL detection

Team no: 5

Team members:

1) Vignesh P (2018103080)
2) Pravin k (2018103050)
3) Bharathmukil R (2018103520)

Date: 06.05.2021

COLLEGE OF ENGINEERING GUINDY - 600025

ANNA UNIVERSITY

Table of Contents

1. INTRODUCTION

Internet and communication facilities faced an enormous growth in the last couple of decades. On the other side the positive growth on internet also lead to the vulnerability of cyberattacks. Attackers use various kinds of techniques to hack a victim's system. Phishing is one of the most common techniques used by the attackers to perform a cyberattack. The no of phishing attacks is increased by 250% per month in the year 2018.

The attacker usually builds a website similar to the target or embeds the exploit code of browser vulnerabilities on the webpage. It tricks the victim into clicking on these links (URL's) to obtain the victim's information or control the victim's computer. The proposed system will detect the malicious links created by the attackers.

1.1 ABSTRACT:

Phishing is the most commonly used technique in cyberattacks. The hacker will create a URL similar to some of the popular benign URL. The victim tends to click these malicious URL links either accidentally or willingly. This might open a gateway for hackers to gain information from the victims system or even provide control over the victims system. Machine learning can be used as a tool to identify such malicious URLs. To detect whether the given URL is malicious or not primarily we should have data of previously created malicious URLs. The data on benign URLs are also analysed to detect malicious URLs. Attackers often utilize the "domain impersonation'' method to trick victims by replacing a letter in a domain name with similar letters. For example, replace 'google' with 'goog1e'. These minor changes cannot be identified by the internet users. The URL is analysed both semantically and visually for most accurate classification.

Current works on detecting malicious URL is described in the chapter 2. The entire system design of the system with module wise split up, dataflow diagram and block diagram is described in chapter 3. Chapter 4 gives information on the experimental result with the system test case. In chapter 5 the result of the system is analysed completely. In the final chapter 6 the future research on malicious URL detection is described with the references.

2. RELATED WORK:

This chapter provides a brief view on previously published research on malicious URL detection. The popular detection methods of malicious URLs mainly include Blacklist, machine learning, rule matching, and deep learning-based detection methods. Kührer [1] analyzed the IP addresses and domain names of multiple public blacklist data sets. They found that parked domains in the Blacklist can constitute a considerable number of blacklist entries, hence, they developed a graph based approach to recognize the sinkholes within the blacklists. An automatic blacklist generator (AutoBLG) was proposed by Sun [2] to automatically generate new malicious URLs based on the original blacklist to expand the original blacklist set. But the performance of the URL detection is limited to the size of the blacklist. To solve this problem, methods have been proposed based on machine learning and deep learning. Vanhoenshoven [3] utilized multiple machine learning algorithms such as K-nearest neighbour, Decision Tree, Naïve Bayes, Random Forests, and Support Vector Machine to detect malicious URLs in datasets with three different statistical characteristics. However, these traditional machine learning algorithms need to manually extract the features of URLs. The feature vectors extracted in this way can only express the shallow features of the data, but not the buried features. Moreover, the methods of manual statistical features will consume a huge deal of time in feature extraction and selection. A deep learning model was proposed by Saxe [4] based on multi-core convolution, expose utilizing a combination of character embedding and convolutional neural networks (CNN) to extract the malicious URLs features and detect the malicious URLs. The experimental results indicated that expose outperforms those manual feature-based models. A malicious domain name detection method was presented by Woodbridge [5] based on long short term memory network (LSTM) utilizing the LSTM model to automatically extract the context features of malicious domain names, and completed the DGAs classification. However, this LSTM model is not very effective in discovering some families with very extraordinary structures. These earlier works on malicious URL detection proves that deep learning algorithms leads to more accurate and better results.

2.2 METHODOLOGY:

Our feature processing has two parts in total. By character embedding and word embedding components, displayable characters and words are embedded into a multidimensional feature space encoding the original URL into a two-dimensional tensor. The image component converts the URL into a grayscale image. Attackers trick victims by replacing a letter in a domain name with similar letters. It is essential to find a way to express the characters and learn hidden information better on the computer to insert the URL into the model for training. One hot vector representation is a basic

vector method. In this method high dimensionality of data is a problem. So displayable characters are embedded into a floating point matrix to better get the hidden information of the letters in URL. Meaningful words in the URL are also mapped to a floating point matrix utilizing word embedding methods. These two matrix are converted to a two dimensional tensor. pe files of the same family have similar texture structures. They converted the malware into grayscale images and used it for classifying it against the CNN network. Within malicious URLs, the URL of the same family or the URL generated by the same tool possesses a similar structure. Therefore, the malicious URL is also converted into images. Each characters are converted to corresponding ASCII values. Then this reshaped into two dimensional matrix. This method can intuitively represent the spatial pattern and structure of URLs.

3. SYSTEM DESIGN:

3.1 BLOCK DIAGRAM:

Our model will detect the malicious URL with two methods. First method uses semantic feature, second method uses visual features. Semantic feature helps to learn hidden information of a URL. All the URLs are converted to one-hot vector, Then these are embedded into a floating point matrix. Attackers are also use meaningful words as developer. Hence, the words in the URL are also mapped to a floating-point matrix utilizing word embedding methods. Then these two matrix are optimized through back propagation. Through this matrix, URL is converted to a two-dimensional tensor.

Within malicious URLs, the URL of the same family or the URL generated by the same tool possesses a similar structure. Therefore, the URLs are converted to grayscale images. This method is easier to identify the spatial similarity of URLs generated by the same family or the same tool. These Images used for classifying against the CNN model.

After the feature extraction, LSTM and CNN are used to train the model. The trained model is evaluated, tested and saved. The convolution neural networks is used to train and save the grayscale model. LSTM is used to train and save the embedded tensor. Then these two model are combined through voting classifier. The models are saved and finally it's time for general uses. The input URL is extracted the feature through the semantic and visual methods. Then this classified through CNN and LSTM model, and predicted as malicious or benign URL. Fig.3.1 gives a clear view on the entire malicious URL detection system.

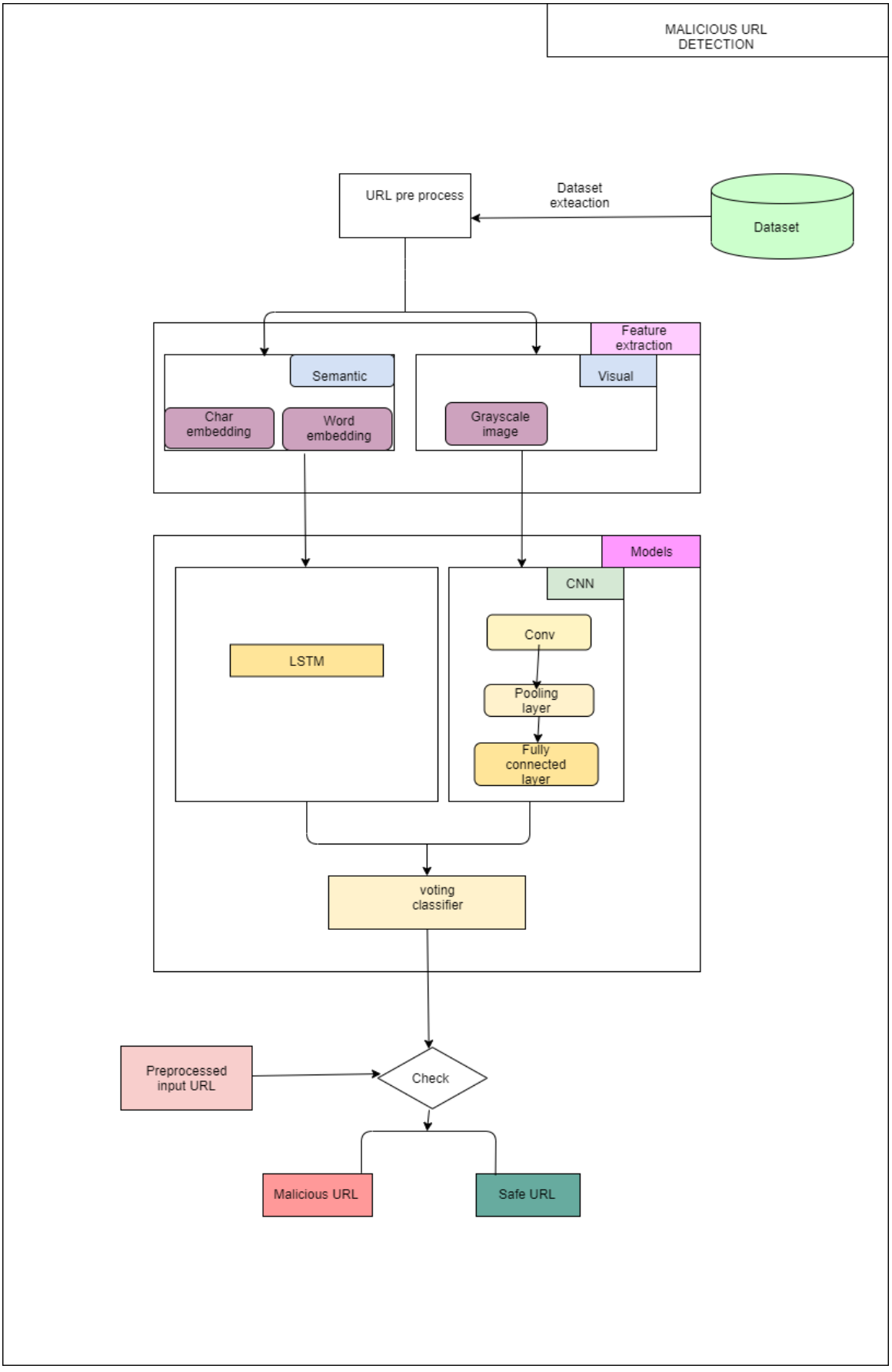


Fig. 3.1 : Block diagram

3.2 Data flow diagram:

Data from the dataset are preprocessed as visual and semantic feature as shown in Fig 3.2. In semantic feature each data is embedded into word and character embedded vectors, then combined into a tensor. In visual feature each data is converted to grayscale image. Then CNN and LSTM are trained simultaneously. CNN trained using grayscale image and LSTM trained using tensor. Then both models are combined and ready for prediction. Then new input URL is preprocessed and predicted using the model.

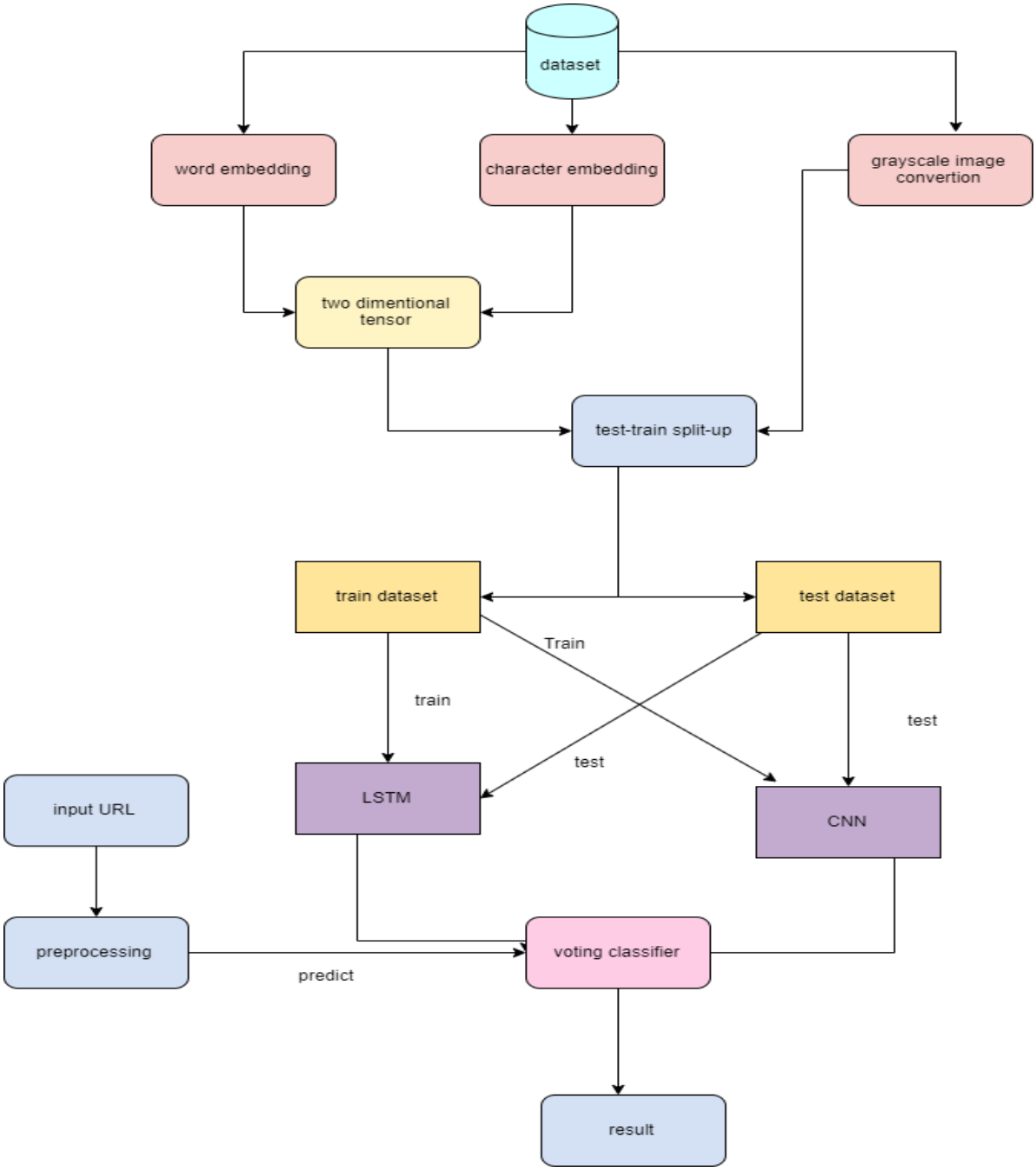


Fig. 3.2 : Data flow diagram

3.3 Module split up:

3.3.1 Module-1:semantic feature extraction

It has two parts. First one is semantic feature. By word embedding components, meaningful words will embedded into a multidimensional feature space encoding the original URL into a floating point matrix. By character embedding components, Displayable characters are embedded into a *s*×*m* floating-point matrix to better get the hidden information of the letters in URL, where s denotes the number of displayable characters, and m refers to the dimension of the embedded vector. Fig.3.3 shows the complete semantic feature extraction process.

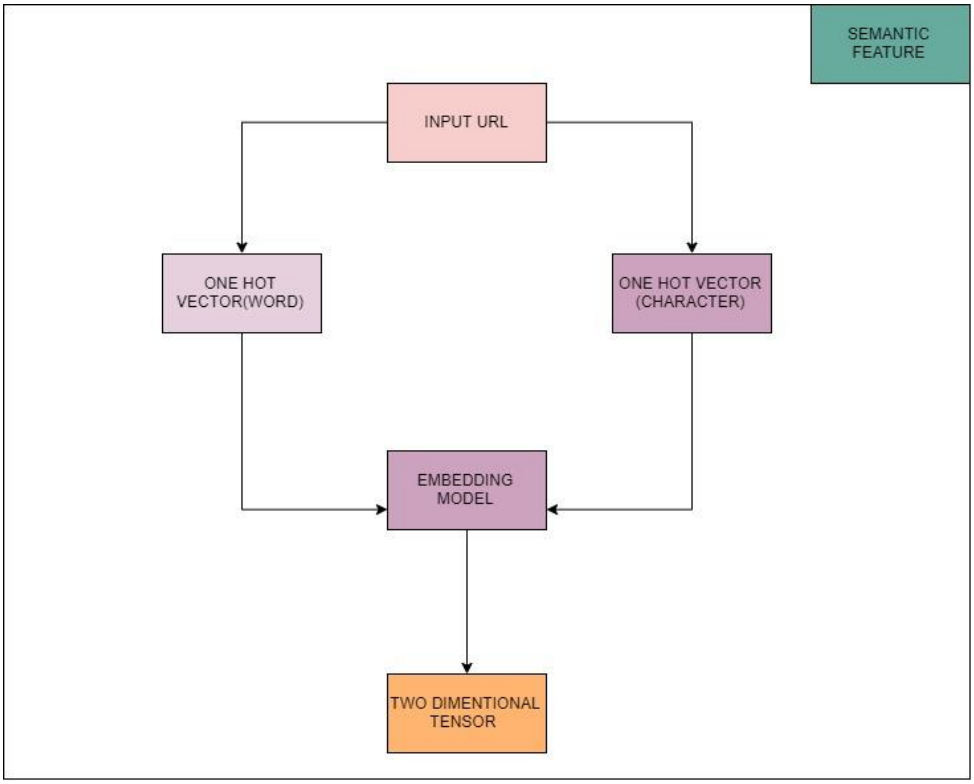


Fig. 3.3 : semantic feature extraction

3.3.2 Module-2: Visual feature extraction:

In this process displayable characters are converted to ASCII values and converted into two dimensional matrix , then visualized into grayscale images. . Fig.3.4 shows the complete visual feature extraction process.

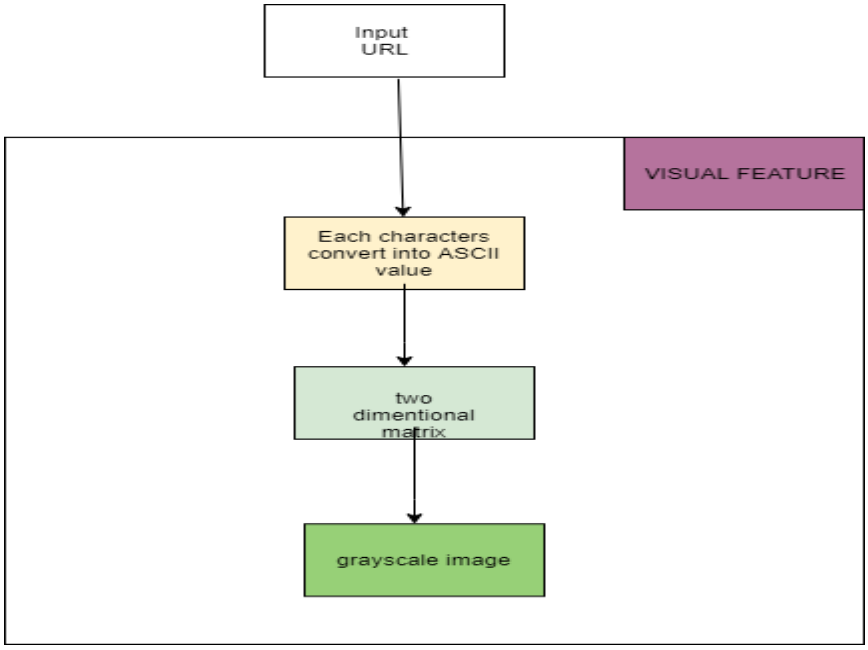


Fig. 3.4 : visual feature extraction

3.3.3. Module -3: Building LSTM model:

In this module, embedded tensor has taken as input and train the model. We used LSTM model with four layers. We also used sigmoid activation function. This model trained and tested with the input images.

Procedure for building the LSTM model:

- Prepare the data
- Feature Scaling (Preprocessing of data)
- Split the dataset for train and test
- Converting features into NumPy array and reshaping the array into shape accepted by LSTM model
- Build the architecture for LSTM network
- Compile and fit the model (Training)
- Evaluate the performance of model(Test)

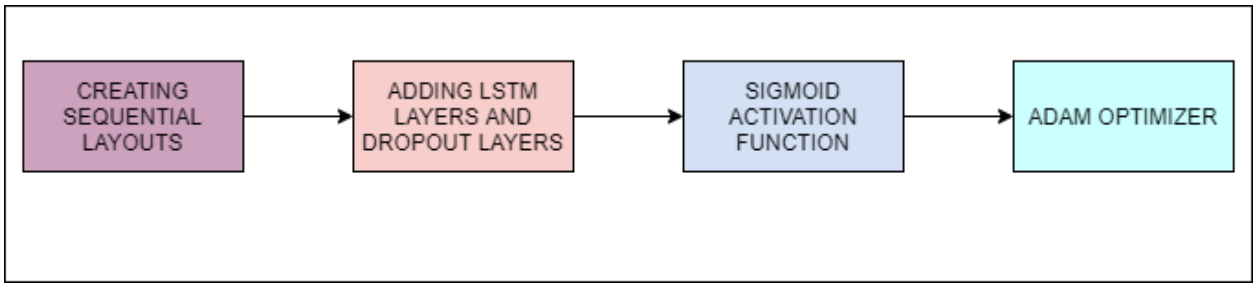


Fig. 3.5 : LSTM model

3.3.4 Module-4: building CNN model

We use a sequential API for model building and create a CNN with layers like convolutional, RelU, Max Pooling, Dropout. We also use softmax function to output a vector. For training the CNN Model, the images in the training set and the testing set are fitted to the sequential model we built using Keras library.

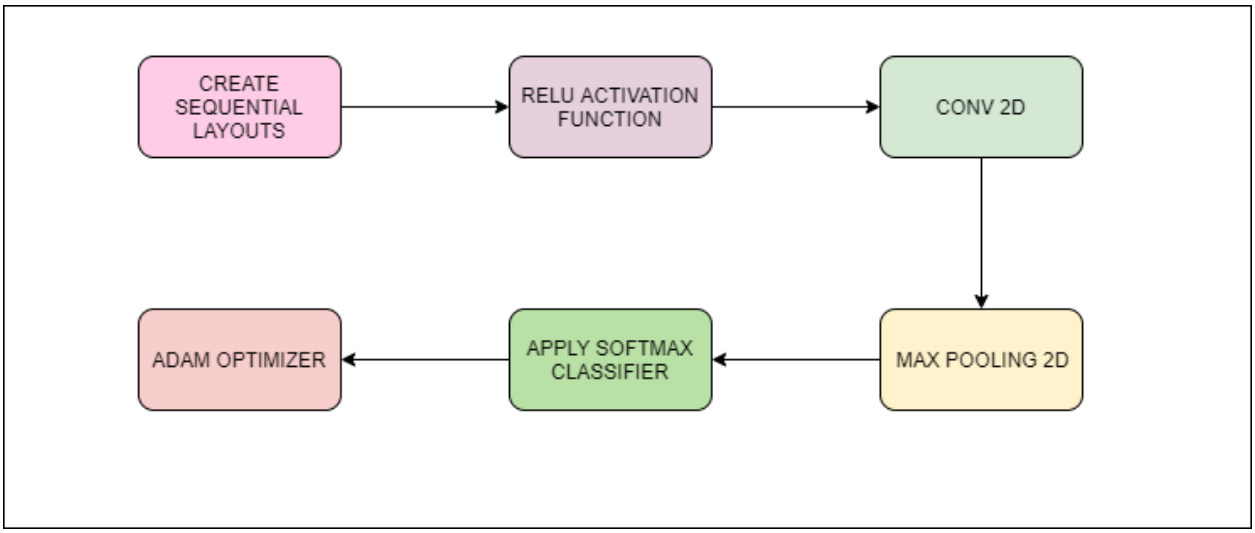


Fig. 3.6 : CNN model

3.3.5 Module-5: voting classifier and predict output

LSTM model and CNN model are combined via voting classifier and built our software. Then the input URL is get from the user, and preprocess for our model, and predict it is malicious or not.

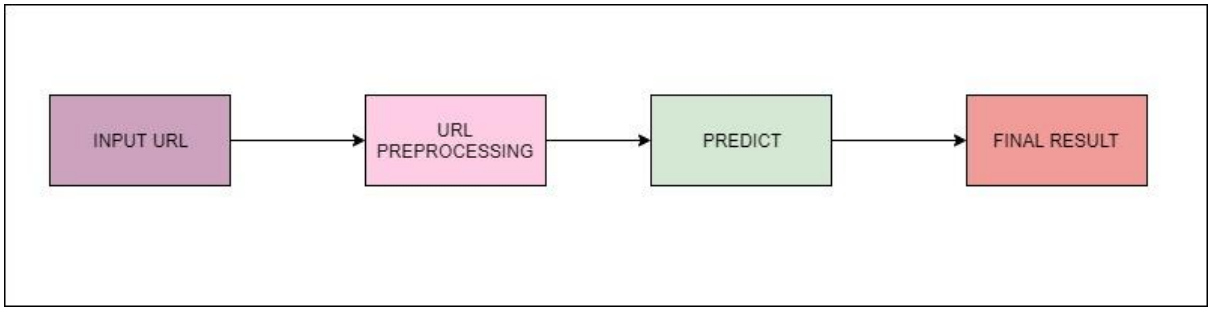


Fig. 3.6 : malicious URL detection.

4. Experimental Results

4.1 DATASET

The set of phishing URLs are collected from open source service called Phis Tank. This service provide a set of phishing URLs in multiple formats like csv, json etc. that gets updated hourly. To download the data: https://www.phishtank.com/developer_info.php. From this dataset, nearly 10000 random phishing URLs are collected to train the models.

The benign dataset was collected through the Alexa website ranking and web search. We acquired a total of 66,017 URLs, of which 32,519 were benign and 33,498 were malicious.

4.2 Experimental results

The CNN model trained with 50 epochs is shown in the fig 4.1.

```
# compiling the sequential model
cnn.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')

# training the model for 10 epochs
history=cnn.fit(cnn_X_train, cnn_Y_train, batch_size=128, epochs=50, validation_data=(cnn_X_test, cnn_Y_test))
```

```
0.9905
Epoch 45/50
125/125 [==============================] - 8s 67ms/step - loss: 0.0016 - accuracy: 0.9999 - val_loss: 0.0531 - val_accuracy:
0.9885
Epoch 46/50
125/125 [==============================] - 8s 67ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.0549 - val_accuracy:
0.9902
Epoch 47/50
125/125 [==============================] - 8s 67ms/step - loss: 0.0012 - accuracy: 0.9999 - val_loss: 0.0590 - val_accuracy:
0.9907
Epoch 48/50
125/125 [==============================] - 9s 70ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.0557 - val_accuracy:
0.9902
Epoch 49/50
125/125 [==============================] - 9s 70ms/step - loss: 8.8473e-04 - accuracy: 1.0000 - val_loss: 0.0624 - val_accura
cy: 0.9902
Epoch 50/50
125/125 [==============================] - 9s 69ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.0552 - val_accuracy:
0.9897
```

Fig. 4.1 : Training CNN model.

The CNN model produces an accuracy up to 98.97% shown in fig.4.2.

```
In [37]: scores = cnn.evaluate(cnn_X_test,cnn_Y_test)
         print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
125/125 [==============================] - 1s 6ms/step - loss: 0.0552 - accuracy: 0.9897
Accuracy: 98.97%
```

Fig. 4.2 : CNN model Accuracy.

The LSTM model trained with 10 epochs is shown in the fig 4.3.

```
In [66]: lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
         history=lstm.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, batch_size=64)

         Epoch 1/5
         250/250 [==============================] - 425s 2s/step - loss: 0.1253 - accuracy: 0.9427 - val_loss: 0.0154 - val_accuracy: 0.
         9960
         Epoch 2/5
         250/250 [==============================] - 440s 2s/step - loss: 0.0047 - accuracy: 0.9991 - val_loss: 0.0140 - val_accuracy: 0.
         9955
         Epoch 3/5
         250/250 [==============================] - 433s 2s/step - loss: 0.0014 - accuracy: 0.9997 - val_loss: 0.0168 - val_accuracy: 0.
         9957
         Epoch 4/5
         250/250 [==============================] - 438s 2s/step - loss: 7.0637e-04 - accuracy: 0.9999 - val_loss: 0.0217 - val_accurac
         y: 0.9955
         Epoch 5/5
         250/250 [==============================] - 434s 2s/step - loss: 4.1960e-04 - accuracy: 0.9999 - val_loss: 0.0285 - val_accurac
         y: 0.9935
```

Fig. 4.3 : Training LSTM model.

The LSTM model produces an accuracy up to 99.27% shown in fig.4.4.

```
In [55]: scores = lstm.evaluate(x_test,y_test)
         print("Accuracy: %.2f%%" % (scores[1]*100))

         125/125 [==============================] - 17s 132ms/step - loss: 0.0491 - accuracy: 0.9927
         Accuracy: 99.27%
```

Fig. 4.4 : LSTM model Accuracy.

Accuracy produced by implementing an ensembling technique called as voting classifier is 99.45% shown in figure 4.5.

```
In [90]: ensemble = VotingClassifier(estimators=models, voting='hard')
         history = ensemble.fit(X_train, y_train)

Epoch 1/7
250/250 [==============================] - 335s 1s/step - loss: 0.1212 - accuracy: 0.9432 - val_loss: 0.0278 - val_accuracy: 0.
9922
Epoch 2/7
250/250 [==============================] - 326s 1s/step - loss: 0.0036 - accuracy: 0.9991 - val_loss: 0.0317 - val_accuracy: 0.
9940
Epoch 3/7
250/250 [==============================] - 325s 1s/step - loss: 7.1017e-04 - accuracy: 0.9999 - val_loss: 0.0349 - val_accurac
y: 0.9945
Epoch 4/7
250/250 [==============================] - 441s 2s/step - loss: 5.5084e-04 - accuracy: 0.9999 - val_loss: 0.0475 - val_accurac
y: 0.9917
Epoch 5/7
250/250 [==============================] - 378s 2s/step - loss: 2.9525e-04 - accuracy: 0.9999 - val_loss: 0.0355 - val_accurac
y: 0.9947
Epoch 6/7
250/250 [==============================] - 399s 2s/step - loss: 6.8101e-05 - accuracy: 1.0000 - val_loss: 0.0383 - val_accurac
y: 0.9945
Epoch 7/7
250/250 [==============================] - 409s 2s/step - loss: 4.1899e-05 - accuracy: 1.0000 - val_loss: 0.0400 - val_accurac
y: 0.9945

In [91]: scores = ensemble.evaluate(x_test,y_test)
         print("Accuracy: %.2f%%" % (scores[1]*100))

125/125 [==============================] - 29s 230ms/step - loss: 0.0400 - accuracy: 0.9945
Accuracy: 99.45%
```

Fig. 4.5 : combined model Accuracy.

4.4 Final Results:

INPUT URL: https://privatbank.ua/ua/business/universalnyje-reshenija/kliring-v-privat24/gtm.start

The URL is correctly classified as benign URL shown in 4.6.

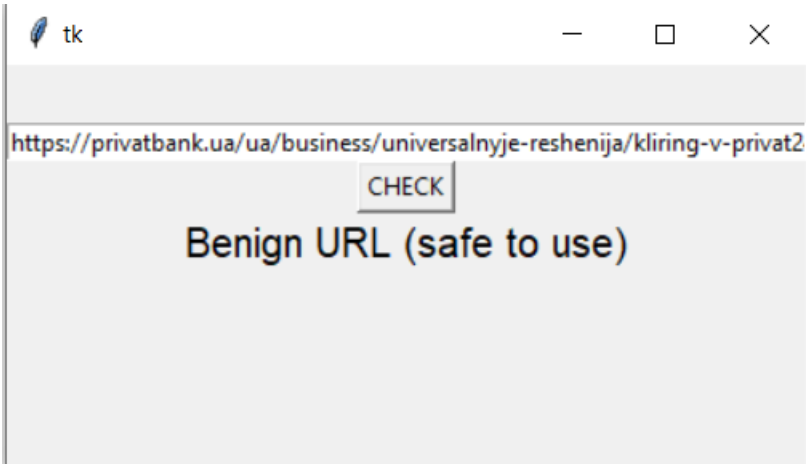


Fig. 4.6 : URL classification.

INPUT URL: https://privatbank.ua/ua/business/Universalnyje-/kliring-v-privat24/gtm.start

The URL is correctly classified as malicious URL shown in 4.7.

Fig. 4.7: URL classification.

5. Result analysis

5.1.1 Evaluation parameters:

1. Accuracy
2. Precision
3. Recall
4. F-value

5.1.2 Formulas for calculating parameters:

1. Accuracy = (TP + TN)/(TP + TN + FP + FN)

2. Precision = TP / (TP + FP)

3. Recall = TP/ (TP + FN)

4. F-value = (2 * P * R) / (P + R)

Where, TP: no of URLs correctly classified as malicious.

TN: no of URLs correctly classified as benign.

FP: no of URLs wrongly classified as malicious.

FN: no of URLs wrongly classified as benign

Classification reports for the model after running 10 epoch, the Presicion, Recall and f1 score along with their support values for 2 main URLs (malicious and benign) listed as 0,1 respectively. The classification report for the LSTM model is shown in fig 45.1

```
In [133]: import sklearn
          print(sklearn.metrics.classification_report(y_test,result))
```

	precision	recall	f1-score	support
0	1.00	0.99	0.99	1984
1	0.99	1.00	0.99	2009
accuracy			0.99	3993
macro avg	0.99	0.99	0.99	3993
weighted avg	0.99	0.99	0.99	3993

Fig. 5.1: Classification report for the LSTM model

```
In [205]: import sklearn
          print(sklearn.metrics.classification_report(cy_test,y_result))
```

	precision	recall	f1-score	support
0	1.00	0.98	0.99	1967
1	0.98	1.00	0.99	2026
accuracy			0.99	3993
macro avg	0.99	0.99	0.99	3993
weighted avg	0.99	0.99	0.99	3993

Fig. 5.2: classification report of CNN model

```
In [92]: result = ensemble.predict(x_test)
         a=convert(result)
         a
         import sklearn
         print(sklearn.metrics.classification_report(y_test,result))
```

	precision	recall	f1-score	support
0	1.00	0.99	0.99	1968
1	0.99	1.00	0.99	2025
accuracy			0.99	3993
macro avg	0.99	0.99	0.99	3993
weighted avg	0.99	0.99	0.99	3993

Fig. 5.2: classification report of combined model

5.1.3 Loss function

Binary Cross-Entropy Loss function:

Binary cross entropy is a loss function that is used in binary classification tasks. These are tasks that answer a question with only two choices (yes or no, A or B, 0 or 1, left or right). Several independent such questions can be answered at the same time, as in multi-label classification or in binary image segmentation. Since the malicious URL detection also comes under binary Classification we use binary cross-entropy for both the models.

Formula:

Loss(y ,z) =z-zy + log(1+e^(-|z|) if (z ≥ 0)

Loss(y ,z) =-zy + log(e^z +1) if (z<0)

Loss(y ,z) = max(z,0) – zy +l og(1+e^(-|z|)

5.2 – Performance metrics

Accuracy Vs Epoch:

A graph between accuracy and epoch is plotted. The accuracy of the model raises and remains almost unchanged after a particular epoch. The accuracy of the model during training and validating comes out to be 100% and 98.97%.

```
In [207]: acc_train = history.history['accuracy']
          acc_val = history.history['val_accuracy']
          epochs = range(1,101)
          plt.plot(epochs, acc_train, 'g', label='Training accuracy')
          plt.plot(epochs, acc_val, 'b', label='validation accuracy')
          plt.title('Training and Validation accuracy')
          plt.xlabel('Epochs')
          plt.ylabel('accuracy')
          plt.legend()
          plt.show()
```

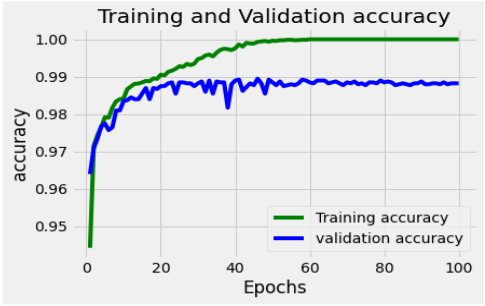


Fig. 5.2: Accuracy vs Epochs.

Loss Vs Epoch:

A graph between Loss and epoch is plotted in fig -----. The Loss in the model lowers and remains almost unchanged after a particular epoch. The Loss in the model is lowered and an exceptional loss less than 4% is achieved

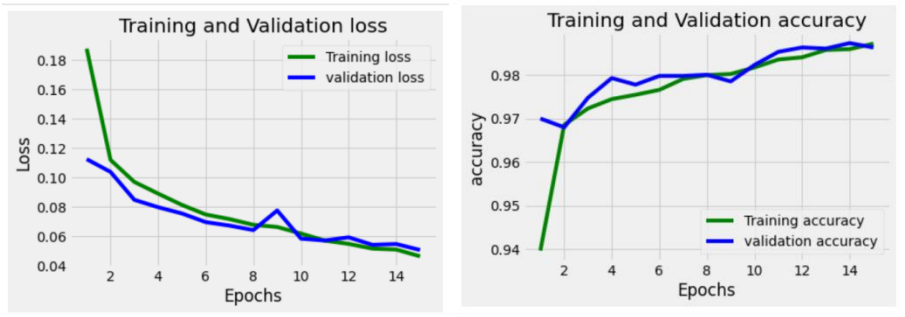
Fig. 5.2: loss vs Epochs

Comparing Models with Accuracy and loss:

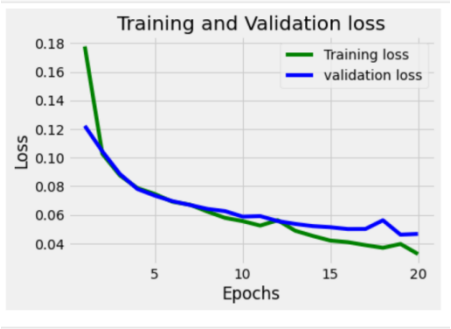
Epoch 10:

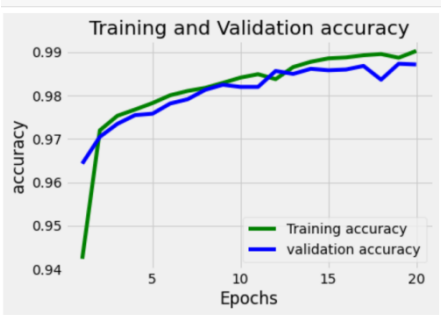
Though accuracy and loss reached a sideway trend after the 5th epoch in *Fig* ---. It was moving up and down in the 14th and 15th epoch giving low accuracy in 14 and high accuracy in 15

Epoch 15:



Epoch 20:





5.3 – Table of inference

Model	Test Size(%)	Validation Size(%)	Epochs	Train accuracy	Test accuracy
CNN	**70**	**30**	**50**	**100**	**98.97**
LSTM	**70**	**30**	**10**	**100**	**99.27**
Voting Classifier	**70**	**30**	**7**	**100**	**99.45**

6. CONCLUSION

6.1 Conclusion

This study explored the possibility of distinguishing benign URLs from malicious URLs by combining the two technologies of CNN and LSTM. To evaluate the proposed method, a dataset was used including over 30,000 malicious URLs from the PhishTank and the malwaredomainlist and more than 30,000 legitimate URLs ranked by Alexa. The proposed method gave a high classification accuracy of 99.45%. Hence, word embedding technology can be used to embed sufficient semantic knowledge in malicious URLs into distributed vectors, then combine with gray images and use neural network models for classification. It only requires some simple processing of the original URL and does not rely on any other complex or expert features. Compared to other methods, our method has a better detection effect. Although the proposed model performs well, further improvement is still

required along with further studies to improve the entire system. We will try to modify our model structure based on malicious classes to perform multiple classifications. In future work, newer better-performing versions may be selected to replace some of these components.

References

1) M. Kührer, C. Rossow, and T. Holz, ``Paint it black: Evaluating the effectiveness of malware blacklists,'' in *Proc. Int. Workshop Recent Adv. Intrusion Detection*, 2014, pp. 1_21.

2) B. Sun, M. Akiyama, T. Yagi, M. Hatada, and T. Mori, ``Automating URL blacklist generation with similarity search approach,'' *IEICE Trans. Inf. Syst.*, vol. E99.D, no. 4, pp. 873_882, 2016.

3) F. Vanhoenshoven, G. Napoles, R. Falcon, K. Vanhoof, and M. Koppen, ``Detecting malicious URLs using machine learning techniques,'' in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1_8.

4) J. Saxe and K. Berlin, ``EXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, _le paths and registry keys,'' 2017, *arXiv:1702.08568*. [Online]. Available: http://arxiv.org/abs/1702.08568

5) J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, ``Predicting domain generation algorithms with long short-term memory networks,'' 2016, *arXiv:1611.00791*. [Online]. Available: http://arxiv.org/abs/1611.00791

6) Y. Kim, ``Convolutional neural networks for sentence classi_cation,'' in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1746_1751.