An Optimized Darknet Traffic Detection System using Modified Locally Connected CNN – BiLSTM Network

Abstract: The contents of the Darkweb have always been a major breach of security and privacy of the general public. Due to its anonymous nature, the detection of traffic from the Darknet becomes difficult. A robust classifier system that accurately predicts and classifies the monitored traffic has become a necessity in this day and age where data privacy is a serious issue. This research work aims to study the effects of the Convolutional-Long-Short-Term Memory (CNN-LSTM) system of classification of Darknet through various deep layer modifications on the nadam optimizer. Experimentations were carried out on different combinations of Locally-Connected CNNs (LcCNN) and Bi-directional LSTM (BiLSTM) to improve the accuracy. The data used to train the model has also been subjected to various levels of Synthetic Minority Oversampling Techniques (SMOTE) to reduce overfitting, data imbalance and provide a better generalization of the data. Additionally, a custom decaying callback function is also implemented. This callback function cuts down the learning rate by half whenever a plateau is reached in the loss value of the model while training. Consequently, It was observed that this decaying callback function tends to improve the accuracy of the model. The results obtained with this implementation in the traffic categorization system outperformed the base CNN-LSTM system with an improved accuracy of 92.57% from 89% using the custom LcCNN-BiLSTM architecture.

Keywords: Darknet, Deep learning, CNN, BiLSTM.

Reference

Biographical notes:

1 Introduction

The Dark web has been the source of illegal activities and transactions since its creation in the 1970s [1]. Owing to its anonymous nature, activities like privacy invasion, sale of restricted goods and mass surveillance have always been rampant on the dark web. This anonymous nature is due to highly encrypted routes and multiple servers the user traverses to keep his identity concealed. This complicated system makes it almost impossible to reproduce the node path and decrypt the information layer by layer of the user.

Darknet is described as an encrypted network that is built on top of the darkweb as part of the internet [2]. Since the network is encrypted, access to the darkweb through the internet requires specialized tools and platforms. Tor, short for "The Onion Router", is one such virtual computer network tool that is used to access the elements of Darknet [3]. Tor acquires access to the Darknet through second-generation onion routing. In this routing, the network circuit is built in increments through multiple servers causing the traffic from the darknet to be usually classified as misconfiguration by the host. Classifying whether the incoming misconfigured traffic is sourced from the darknet is one of the challenges faced by the IT and cybersecurity sectors.

The significance of monitoring and classifying this "misconfigured" or darknet traffic is to provide comprehensive details about possible threats or vulnerabilities for further investigation. Since Tor plays a salient role in accessing the darknet, researchers are focusing on analyzing and monitoring the traffic incoming from Tor applications and classifying them for malicious activities. This justifies the need of a robust classification model for darknet traffic which accurately segregates Tor traffic from Non-Tor, VPN (Virtual Private Network) and Non-VPN traffic.

In this research work, the effects of Locally-Connected Convolutional Neural Networks (LcCNN) and Bi-directional Long Short Term Memory (BiLSTM) is investigated on the CIC-Darknet2020 dataset [4] and additionally the previous classification system [5] has been improved by modifying the deep layers of the architecture. The first step in creating this system was to solve the data imbalance that arises in the Darknet dataset. Synthetic Minority Oversampling Techniques (SMOTE) [6] sampling method has been implemented to overcome the data imbalance and other overfitting issues that may arise. Effective set of features were selected using XGBoosting and Principal Component Analysis (PCA) analysis for ideal model performance [7, 8]. Various multi-class classifier neural network architectures were implemented. Unlike the past system where the Conventional CNN-LSTM was used, LcCNNs [9] has been used. The LcCNNs layer works similar to the CNN layer, except that the weights are unshared in LcCNNs. A different set of filters are applied

at every different patch of the input. These are further coupled with BiLSTMs [10] which are nothing but two LSTMs [11] where one works in the forward direction and the other works in the backward direction. This combination of LcCNNs and BiLSTMs, increased the performance of the model by 4.5% from the original system.

To further improve the accuracy, the Adam optimizer has been traded with Nadam optimizer [12] as it performed better compared to the former. A custom decaying callback function is implemented that reduces the learning rate by half once a plateau is reached in the loss value of the model while training. This ensures local minima are avoided and the loss value reaches close to the global minima. The callback function also keeps track of the best weights during training. Rigorous experimentations were conducted by modifying the deep layers of the architecture and tuning the hyper parameters that proves the efficacy of the proposed architecture in better classification of darknet traffic.

The paper is structured as follows: Section 2 focuses on the related work and contributions. A brief description of the dataset used is given in Section 3. System core and design are elaborated in Section 4. Experimental analysis and results are discussed in Section 5. Section 6 concludes the paper.

2 Related Work and Contributions

This section gives a brief overview of the past work based on Darknet datasets and investigation into the significance of various features for classifying different traffic types.

In this research work an attempt has been made to build upon the system proposed by Sarwar et al. [5]. They proposed their best CNN-LSTM architecture which was trained on a dataset having its best features extracted using XGBoosting. They obtained precision and recall score of 0.90 and 0.88 respectively. The data was also subjected to SMOTE [6] on the Tor class by 20% for minimizing the data imbalance.

Demertzis et al. [13] implemented fifteen different classifier techniques on this dataset ranging from decision trees, k - means classifier to XGB classifier with the highest accuracy of 90%. Their work on the reservoir network further increased efficiency and inference speed on noisy scattered data that previous classification methods could not handle. Although the recall score lacked, this work demonstrated the effects of reservoir network implementation on this dataset.

Ali et al. [14] worked on 80/TCP and 53/UDP packets classification. They adopted Resource Allocating Network with Locality Sensitive Hashing (RAN-LSH) in which data to be trained are selected by using LSH and fast online learning is actualized by training only selected data. This resulted in an accurate backscatter DDoS detector.

Another implementation by Aswad and Sonuç [15] on Spark engine to prevent unnecessary pre-processing of the dataset presented the first implementation of ANN accompanied by Apache spark engine which outperforms the autoencoder method on the CIC-Darknet2020 data.

Vishnupriya et al. [16] provided the effects of recurrent neural networks and Deep Neural Network on the CIC-Darknet2020 dataset. This work concluded that the DNNs provided large numbers of false positives and false negatives and the LSTM architecture provided better accuracy and precision.

Iliadis and Kaifas [17] used k – nearest neighbour, gradient boosting, decision trees, random forests and a multilayer perceptron model to identify darknet traffic. The results and evaluation metrics in this paper portrayed that Random Forest performed the best out of all machine learning algorithms that were tested.

The work in Berman and Paul [18] signifies the classification of text and images from Darknet and Wikipedia where the text might not be as descriptive. The authors demonstrated that the CNN and Latent Dirichlet Allocation(LDA) models in conjunction can retrieve documents and images based on both text and image queries.

Basher et al. [19] presented an extensive work of Web and P2P traffic using full packet traces collected at a large edge network. They consider three flow-level metrics, namely flow size, flow IAT, and flow duration, and three host-level metrics, specifically flow concurrency, transfer volume, and geographic distance. And observed a contrast in features between Web and P2P traffic.

Arlitt and Carlsso's [20] work examined the communication patterns of three groups of external organisations: first-party providers, third-party providers, and ISPs.It was portrayed how scanning a network can reveal a lot of information about the edge network under surveillance.The authors also further investigated DDos filters and how DDos attacks bypassed these filters by launching the attacks from source ipv4 addresses.

Keeping in sight of the above systems, this paper aims to make the following contribution.

- Showcase the model performance on 20% SMOTE sampling on the Tor alone and 100% SMOTE sampling on all classes for better generalization of the model.
- Implement a custom LcCNN [9] and BiLSTM [10] based architecture for better accuracy and recall score.
- Provide a custom decaying callback function to improve the precision of the model.
- Fine-tune hyperparameters to improve inference speed and loss value of the model.
- Present comparative analysis of the effect of each modification to the existing system.

3 Dataset Description

The dataset selected for this system is CIC-Darknet2020 published by Canadian Institute for Cybersecurity [4]. This dataset is an amalgamation of two pre-existing public datasets namely ISCXTor2016 [21] and ISCXVPN2016 [22]. Data was captured using Whonix, a ready-to-use Linux OS configured to route all traffic through the Tor network. The Whonix distribution consists of two virtual machines, the gateway and the workstation. This system connects to the Internet through a gateway virtual machine, which in turn routes all the traffic through the Tor network. With this configuration, using the Tor network at the workstation virtual machine, it becomes transparent that the incoming traffic is indeed Tor traffic. The authors captured the outgoing traffic at the workstation and the gateway simultaneously, one for regular and one for Tor and labelled them according to the type of traffic and application of the traffic.

The dataset consists of two layers: first layer defines the benign and Darknet nature of the traffic.This consists of 134,348 traffic data samples which are benign and 24,311 which are sourced from darknet. The second layer signifies the application of the incoming traffic and consists of app categorization from classes like Audio-Stream, Browsing, Chat, Email, P2P, Transfer etc. The dataset consists of 85 features which include two types of label category namely, Layer 1's traffic type: Tor, Non-Tor, VPN, Non-VPN; Layer 2's app category: Audio-Stream, Browsing, Chat, E-mail, P2P, Transfer, Video-Stream and VoIP. Table 1 provides the distribution of the data samples of each class in Layer 1 and Table 2 provides the distribution of the data samples of each class in Layer 2.

Table 1 Distribution of the data samples of each class in Layer 1 Traffic Type

TRAFFIC CATEGORY	SAMPLES
Non-Tor	93357
Tor	1393
Non-VPN	23864
VPN	22920

Table 2 Distribution of the data samples of each class in Layer 2 App Category

TRAFFIC APPLICATION	SAMPLES
Audio Streaming	18065
Browsing	32809
Chat	11479
E-mail	6146
File Transfer	11183
P2P	48521
Video Streamin	9768
VOIP	3567

4 System Core and Design

Pre-processing of data samples and feature extraction processes are briefly reviewed in subsection 4.1. Classification methods and the proposed custom Neural network architecture are described in subsection 4.2.

4.1 Pre-Processing and Feature Extraction

For data cleaning and filtering, the data was first subjected to 'drop duplicates' and 'drop N/A' value functions using the Pandas library based on Python 3. Post this, data was analyzed using data exploratory tools like the Matplotlib and Seaborn library. Consequently, data imbalance was observed in the dataset which was due to the very low numbers of data samples in the Tor class.

4.1.1 SMOTE

To conquer this data imbalance, SMOTE [6] was implemented at various sampling levels on the data set using the Keras library. SMOTE is a technique used to procedurally generate new data samples by selecting a data sample in a class and its k nearest neighbours in the same feature space. Once the k neighbours are assigned, one neighbour is randomly selected and a new data is generated in a random spot between the neighbour and the data sample having the same feature space. The dataset was initially subjected to SMOTE such that there is an increase of data samples in the imbalance class (Tor) by 20%. This 20% sampling was performed to replicate and compare the results as obtained in [5] where the authors had a similar approach with SMOTE. But contributing to the scarcity of the Tor data samples and how the test train split will be executed, the Tor data samples would be even scarce and not generalised properly in the test set. So, If the model correctly classifies all the classes except the tor class in the test set, it would represent a minor drop in accuracy since the number of samples for tor dataset is less. To resolve this, SMOTE was implemented across all classes uniformly such that all classes had an equal number of data samples. Consequently, increasing the total data size from 1,16,711 to 2,37,226 samples. These samples were uniformly distributed across the dataset and provided maximum generalization in all classes. However, to maintain a realistic accuracy of the model, the model was trained on the SMOTE generated data and tested on the original data samples.

4.1.2 PCA and XGBOOSTING

Once the sampling and pre-processing is completed, selected features need to be extracted from the dataset to reduce redundancy. Feature extraction also ensures that fewer resources are required by the model while training and increases the generalization steps in the machine learning process. Data was subjected to two feature extraction methods – PCA and XGBoosting

[7, 8]. Although the dataset was over-sampled, the top 20 features obtained were the same that were observed by the authors in [5]. The top 20 features selected using XGBoosting are shown in the Table 3.

Table 3 Top ranked features

Rank	Feature
1	Idle Max
2	FWD Init Win Bytes
3	Idle Mean
4	Idle Max
5	Fwd Seg Size Min
6	Subflow Fwd Packets
7	Flow Duration
8	Flow IAT Max
9	Flow IAT Min
10	Flow IAT Mean
11	Fwd Packets/s
12	Flow Packets/s
13	Bwd Init Win Bytes
14	Protocol
15	FIN Flag Count
16	Bwd Packets/s
17	Fwd IAT Max
18	Bwd Packet Length Mean
19	Bwd Packet Length Min
20	Fwd IAT Total

4.2 Classification systems

Post pre-processing and feature extraction, data was subjected to various neural network layers and architecture to create a model that expresses the highest precision and least loss value.

4.2.1 CNN and LSTM layer

In the previous CNN-LSTM system [5], the CNN layer was utilized for feature extraction and the LSTM supported the sequence prediction. The following equations (Eqs. (1) and (2)) give the main computing logic of the neural network where series input is denoted by x_t, series output is denoted by y_t, hidden memory cells are denoted by h_t, weight matrices are denoted by W, and bias vectors are denoted by b.

$$h_t = H(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \tag{1}$$

$$P_t = W_{hy}y_{t-1} + b_y \tag{2}$$

Since the filter weights are shared between all samples in a traditional CNN layer, the layer tends to lose out on the subtle features in the dataset. This issue can be rectified using LcCNNs [9], where each sample data is assigned to a different unshared filter. This enables the model to learn from each feature in the data sample with a different aspect, capturing the hidden or minute features in the dataset. Furthermore, the sequence prediction is also enhanced by implementing a BiLSTM [10] which consists of two LSTM units; one in the forward direction and the other in the backward direction. The resulting LcCNN-BiLSTM architecture has better precision due to the compounded effect of LcCNNs and BiLSTMs which enable additional training by traversing the input data twice i.e left-to-right and right-to-left [23].

4.2.2 Nadam Optimizer

Another fine-tuning method that was adopted to stabilize the loss value was the implementation of the Nadam Optimizer [12]. Nadam, unlike the Adam Optimizer used in the past system, is in essence, similar to the Adam optimizer with the key difference being RMSprop with Nesterov momentum instead of RMSprop with base momentum in the Adam optimizer.

The Nadam optimizer [12] is defined as follows where t is time step, β is a constant factor, w is weight parameter to be updated, α is the learning rate, v is the cumulative sum of current and past squared gradients, m is momentum and $\frac{\partial L}{\partial w_t}$ is gradient of L, the loss function to minimise, w.r.t L.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon}\left(\beta_1\hat{m}_t + \frac{1-\beta_1}{1-\beta_1^t} \cdot \frac{\partial L}{\partial w_t}\right) \tag{3}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{4}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{5}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t} \tag{6}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\partial L}{\partial w_t}\right]^2 \tag{7}$$

4.2.3 Custom Callback Function

Finally, to improve the accuracy of the model a custom decaying callback function is implemented. This callback function monitors for plateaus for every 5 epochs with a minimum difference in the validation accuracy of 0.01%. If this threshold change remains uncrossed, it is considered as a plateau and the callback function cuts down the learning rate by half. This method ensures peak model training times, avoids local minima and obtains the highest precision possible. This callback function was implemented using Tensorflow's [24] "ReduceLROnPlateau" method.

5 Experimental Analysis and Results

Experimentations were conducted to accurately detect the traffic category of the second layer of the CIC Darknet2020 dataset [4]. The experiments are first

categorized based on the degree of SMOTE on the data, i.e., 20% on the imbalanced class and sampling on all classes. The models are then subjected to training with various configurations and add-ons to the LcCNN-BiLSTM model on the sampled data. Once the training is completed, the models are evaluated using recall score, precision and average loss value of the various configured models. The model was experimented on various hyperparameters and the best performance was recorded in the following set as given in Table 4.

Table 4 Parameters and Values

Parameter	Value
Hidden Layer Activation Function	Relu
Output Layer Activation Function	Softmax
Loss Function	Categorical Crossentropy
Batch size	8192
Saved model	Best weights
Epochs	200

5.1 Hardware and Software

The model was trained on an Intel processor i5 having 16 GB RAM and a GTX 1060 MAX-Q Graphics card. The final system was implemented using Jupyter notebook based on Python 3.9.0. The data was preprocessed and analyzed using NumPy, Matplotlib, and Pandas libraries.

The feature extraction using PCA and XGBoosting was implemented using Keras's predefined methods and finally, these features were trained on the Tensorflow 2.2.0 architecture [24].

5.2 Training and validation

Once the top features are extracted from the dataset, the data is shuffled and split into train and split ratio of 80% train and 20% test. This data split is then experimented with a series of neural networks having different configurations and hyperparameters. This is done in order to perform a comparative analysis of the performance change for each neural component of the model. The first change we make is in the dataset, i.e, we train these models on different SMOTE configurations. We call the SMOTE sampling on only imbalance classes as "partial SMOTE" and sampling on all classes as "Full SMOTE". These generated datasets are then trained on different configurations of CNNs/LcCNNs, LSTM/BiLSTM and the Adam/Nadam optimizers. A decaying learning rate callback function based configuration is also added to the final configured model to ensure global minima is reached for the loss value.

After rigorous hyperparameter tuning, the final models are trained as per the best parameters provided in Table 4. The corresponding training/validation accuracy and losses with respect to epochs are plotted in Figs. 1, 2, 3 and 4. Dataset trained using Adam optimizer on partial SMOTE dataset and Full SMOTE dataset are depicted in Figs. 1 and 2 respectively. Similarly, datasets trained using Nadam optimizer on partial SMOTE dataset and Full SMOTE dataset are depicted in Figs. 3 and 4 respectively.

Comparing Figs. 1a and 1b, it is observed that the introduction of LcCNNs achieves higher accuracy and converges faster as compared to base CNNs 1a. This conveys a more stable and robust learning pattern.

Moving on to the next change in the Neural layers, we can observe erratic changes in loss values in Fig. 1c contributing to the nature of BiLSTMs (to regularly overfit a model). This overfitting effect is mitigated by the custom callback function as shown in Fig. 1d. The custom callback function achieves this by reducing the learning rate by half whenever a plateau is obtained in the loss values whilst training. By reducing the learning rate, the model does not overshoot the loss global minima. Since the learning rate is only decayed when a plateau is reached, the model also successfully avoids local minima. We infer this increase in accuracy across the models from Figs. 1a to 1d. Furthermore, the Figs. 1e, 1f, 1g and 1h denote the loss values of the correspond accuracies portrayed by Figs. 1a, 1b, 1c and 1d respectively. This representation remains the same through Fig. 1 through Fig. 4.

A similar trend is noticed in Fig. 2 apart from the fact that there is not much deviation between training and validation accuracy. This is represented by the overlap in the red and blue lines in Fig. 2 whereas in the partial SMOTE dataset the deviation is considerably more between the two lines Fig. 1. This change in overlap is contributed to the way our model is split into test and train. Since the test size of the model is 20%, The imbalance class has fewer samples in the test set than other classes. This alters the test accuracy of the model as the model makes fewer incorrect predictions on an already small class of samples, Consequently increasing its test accuracy as seen in Fig. 1. Thus equal sampling the dataset in the Full SMOTE dataset rectifies this data imbalance issue in the test set and shows a much more balanced and accurate portrayal of the accuracy of the model on all the classes. The drop in accuracy justifies that the classes were sufficiently balanced and generalised in the test set so as to detect all possible incorrect predictions.

Furthermore, the training process in Figs. 3 and 4 displayed a slow gradual learning curve brought upon by the Nadam optimizer as compared to the tighter learning curve by the Adam optimizer observed in Figs. 1 and 2. These gradual learning curves signify the slower "annealing" of the learning rate by the Nadam Optimizer. This justifies an increase in performance of the model as it avoids local minimas and increases the accuracy by reaching global minima loss.

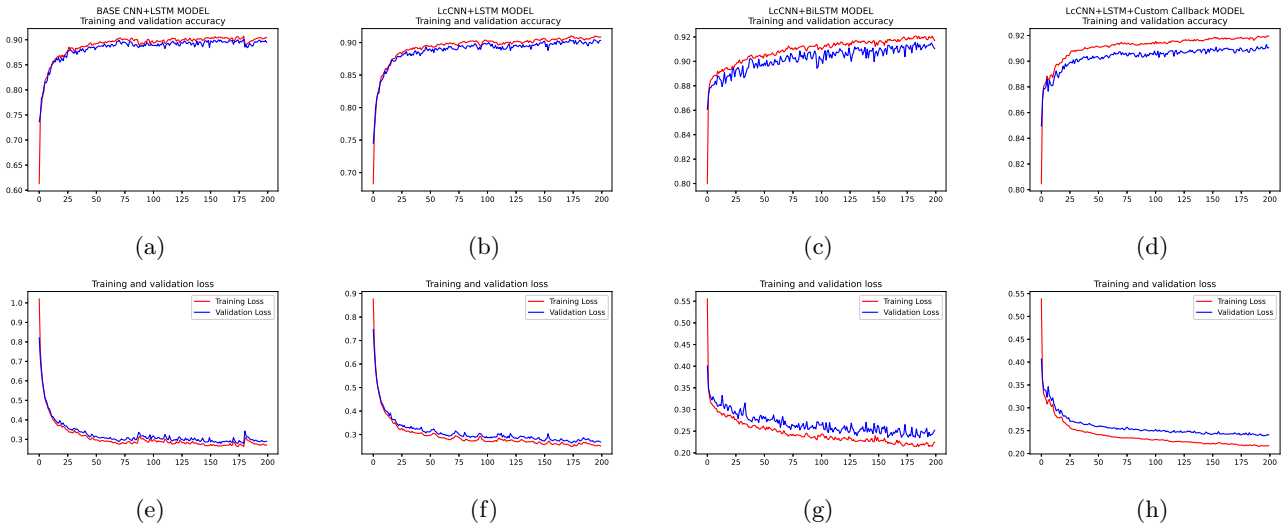

Figure 1: Dataset trained using Adam Optimizer on Partial SMOTE dataset

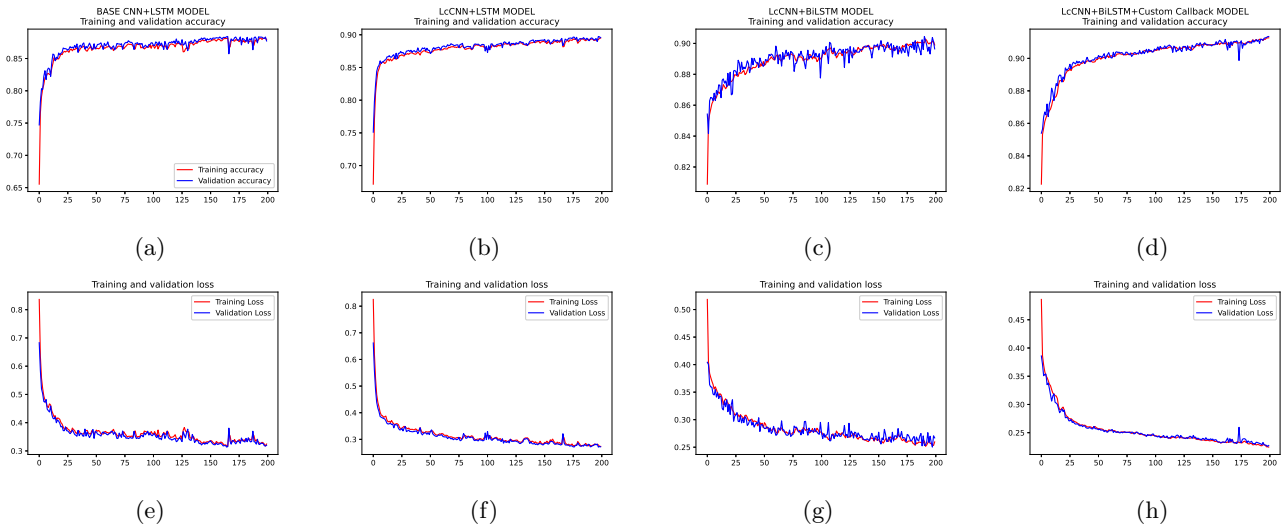

Figure 2: Dataset trained using Adam Optimizer on Full SMOTE dataset

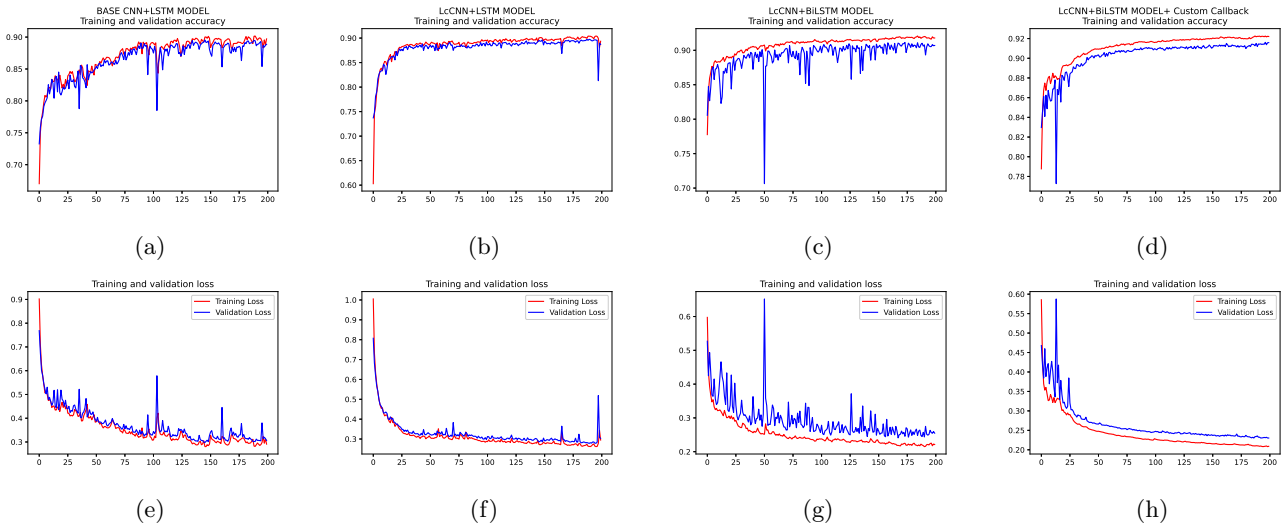

Figure 3: Dataset trained using Nadam Optimizer on Partial SMOTE dataset

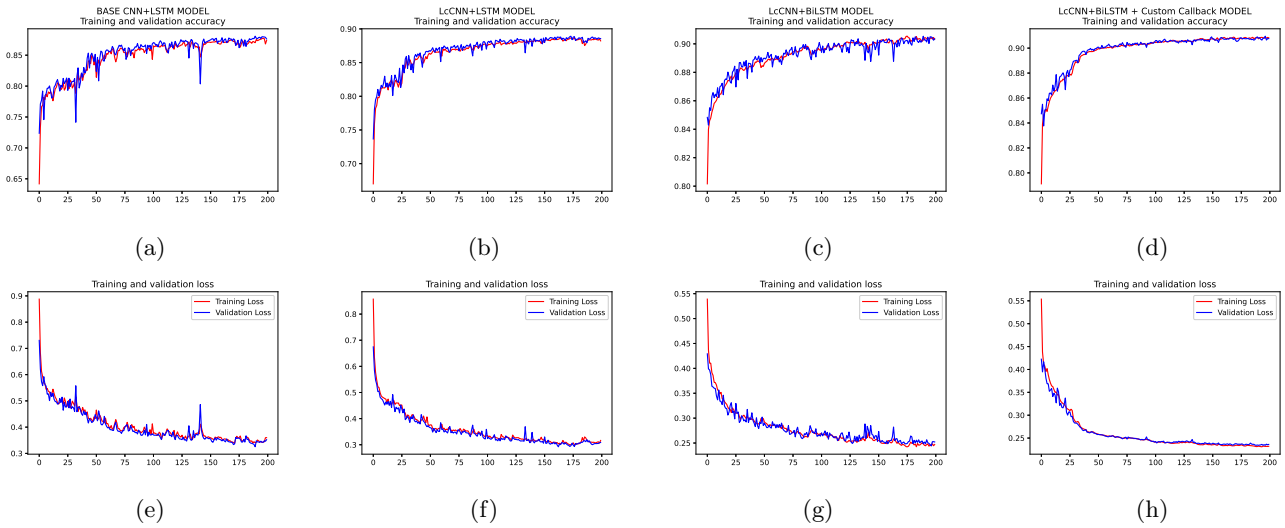

Figure 4: Dataset trained using Nadam Optimizer on Full SMOTE dataset

5.3 Observations

After the training process has been terminated for all the models the following observations are made. The model implemented by [5] was replicated in the base CNN-LSTM model using the partial SMOTE data. This is represented in the first column under the 20% split configuration model in Table 5. From Table 5, it is evident that there is a gradual increase in performance with the addition of LcCNN, BiLSTM and the custom callback function. In the Nadam optimizer section in Table 5, although the Nadam optimizer is unable to reach the highest precision in the partial SMOTE dataset, it increases the overall stability and average recall score of the model and consequently decreases the loss value. Furthermore, in the lower half of Table 5, it is observed that the Nadam optimizer performed much better on the full SMOTE dataset alongside the implementation of the BiLSTM and the custom callback function as presented in the last two columns of Table 5. An increase of 2.77% and 2.5% in the recall and precision value was obtained by comparing the Base CNN + LSTM model and LcCNN + BiLSTM + Callback function using Nadam optimizer in Table 5. A reduction of the average loss value by 0.08 was also observed from the partial SMOTE dataset as utilized by the past system [5]. Even though the final model trained on the partial SMOTE data has an increased precision by almost 2.5%, the model still suffered from data imbalance in the test dataset. The full SMOTE dataset provided a much more balanced prediction on all classes, despite the loss in accuracy.

6 Conclusion

In this research work, we have analyzed the performance of different configurations of the LcCNN – BiLSTM models and fine tuned the hyperparameters over the classification of layer 2 of the Darknet dataset. The resulting final model provided an accuracy of 92.6%, i.e., an increase of 2.8% and a considerable dip in the loss value from the base CNN-LSTM model. This system also mitigates the data imbalance that arises in the dataset by studying the accuracy of the models of various levels of SMOTE sampling of the dataset. From these varying levels of sampling on the dataset, we traded off overall accuracy for a better generalization and realistic performance of the model. Further enhancements can be made in the system using an ensembled or federated learning approach. Although different configurations of the models ensure maximum generalization of the weights and test accuracy, further fine-tuning the Neural network architecture and introducing new data preprocessing techniques can improve the overall accuracy of the system.

References

[1] Julian Broséus, Damien Rhumorbarbe, Marie Morelato, Ludovic Staehli, and Quentin Rossy. A geographical analysis of trafficking on a popular darknet market. *Forensic science international*, 277:88–102, 2017.

[2] Carl-Maria Mörch, Louis-Philippe Cote, Laurent Corthesy-Blondin, Léa Plourde-Léveillé, Luc Dargis, and Brian L Mishara. The darknet and suicide. *Journal of affective disorders*, 241:127–132, 2018.

[3] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[4] Arash Habibi Lashkari, Gurdip Kaur, and Abir Rahali. DIDarknet: A contemporary approach

Table 5 Comparative Summary of different configurations of models

Final Results		Adam Optimizer				Nadam Optimizer			
		Base CNN+ LSTM model	LcCNN + LSTM model	LcCNN+ BiLSTM Model	LcCNN+ BiLSTM+ Custom Callback function	Base CNN+ LSTM model	LcCNN + LSTM model	LcCNN+ BiLSTM Model	LcCNN+ BiLSTM+ Custom Callback function
20% SMOTE Dataset	Recall	88.13% [5]	88.9%	90.37%	90.9%	87.1%	88.15%	89.33%	90.10%
	Precision	90.1%[5]	90.5%	91.72%	92.57%	89.96%	90.22%	91.54%	91.55%
	Average loss	0.32	0.30	0.26	0.24	0.35	0.32	0.29	0.27
100% SMOTE Dataset	Recall	86.27%	87.68%	89.01%	89.88%	86.43%	87.25%	89.46%	90.06%
	Precision	88.44%	89.34%	90.50%	90.89%	88.70%	88.86%	90.83%	91.05%
	Average loss	0.38	0.32	0.28	0.26	0.36	0.34	0.27	0.25

to detect and characterize the darknet traffic using deep image learning. In *2020 the 10th International Conference on Communication and Network Security*, pages 1–13, 2020.

[5] Muhammad Bilal Sarwar, Muhammad Kashif Hanif, Ramzan Talib, Muhammad Younas, and Muhammad Umer Sarwar. Darkdetect: Darknet traffic detection and categorization using modified convolution-long short-term memory. *IEEE Access*, 9:113705–113713, 2021.

[6] Alberto Fernández, Salvador Garcia, Francisco Herrera, and Nitesh V Chawla. Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research*, 61:863–905, 2018.

[7] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[8] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[9] Yu-hsin Chen, Ignacio Lopez-Moreno, Tara N. Sainath, Mirkó Visontai, Raziel Alvarez, and Carolina Parada. Locally-connected and convolutional neural networks for small footprint speaker recognition. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 1136–1140. ISCA, 2015.

[10] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. The performance of LSTM and BiLSTM in forecasting time series. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3285–3292. IEEE, 2019.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] Timothy Dozat. Incorporating nesterov momentum into adam. In *International Conference on Learning Representations Workshop (ICLR Workshop)*, pages 1–4, 2016.

[13] Konstantinos Demertzis, Konstantinos Tsiknas, Dimitrios Takezis, Charalabos Skianis, and Lazaros Iliadis. Darknet traffic big-data analysis and network management for real-time automating of the malicious intent detection process by a weight agnostic neural networks framework. *Electronics*, 10(7):781, 2021.

[14] Siti Hajar Aminah Ali, Seiichi Ozawa, Tao Ban, Junji Nakazato, and Jumpei Shimamura. A neural network model for detecting ddos attacks using darknet traffic features. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2979–2985. IEEE, 2016.

[15] Salma Abdullah Aswad and Emrullah Sonuç. Classification of vpn network traffic flow using time related features on apache spark. In *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pages 1–8. IEEE, 2020.

[16] VishnuPriya. A, Hiran Kumar Singh, SivaChaitanyaPrasad. M, and JaiSivaSai. G. Rnn-lstm based deep learning model for tor traffic classification. *Cyber-Physical Systems*, 0(0):1–18, 2021.

[17] Lazaros Alexios Iliadis and Theodoros Kaifas. Darknet traffic classification using machine learning techniques. In *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–4. IEEE, 2021.

[18] Alexander Berman and Celeste Lyn Paul. Making sense of darknet markets: Automatic inference of semantic classifications from unconventional multimedia datasets. In *International Conference on Human-Computer Interaction*, pages 230–248. Springer, 2019.

[19] Naimul Basher, Aniket Mahanti, Anirban Mahanti, Carey Williamson, and Martin Arlitt. A comparative analysis of web and peer-to-peer traffic. In *Proceedings of the 17th international conference on World Wide Web*, pages 287–296, 2008.

[20] Martin Arlitt, Niklas Carlsson, Phillipa Gill, Aniket Mahanti, and Carey Williamson. Characterizing intelligence gathering and control on an edge network. *ACM Transactions on Internet Technology (TOIT)*, 11(1):1–26, 2011.

[21] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of tor traffic using time based features. In *ICISSp*, pages 253–262, 2017.

[22] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pages 407–414, 2016.

[23] Tao Chen, Ruifeng Xu, Yulan He, and Xuan Wang. Improving sentiment analysis via sentence type classification using bilstm-crf and cnn. *Expert Systems with Applications*, 72:221–230, 2017.

[24] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.