Malicious Network Traffic Classification Using Deep Neural Networks

CSEC620 - Project 1 - Mehul Sen

This project uses a deep neural network to classify network traffic into benign and malicious categories. The model can detect various types of attacks, such as denial of service, distributed denial of service, brute force, botnet, web attacks, and port scans. It is trained on a dataset labeled CICIDS2017, which contains network traffic data captured over five days. The project demonstrates how deep learning can detect and classify malicious network traffic.

Quick Start

This section will outline the necessary steps to begin training and testing the model. This project was completed using Visual Studio Code with Jupyter Notebook. If you require assistance, we recommend referring to the following resource for setting up the notebook: Running Jupyter Notebook on Visual Studio Code | Medium

1. *Setup Dataset Folder*: To set up the dataset folder, start by unzipping and extracting the .csv files from within the `MachineLearningCSV.zip` into a folder named Dataset. If using the provided zip file, `project_1_sen.zip`, the dataset should already be correctly located. The desired folder structure should be as follows with the required files:

```
- requirements.txt
- project_1_malicious_network_analysis.ipynb
- Dataset/
    - Monday-WorkingHours.pcap_ISCX.csv
    - Tuesday-WorkingHours.pcap_ISCX.csv
    - Wednesday-workingHours.pcap_ISCX.csv
    - Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv
    - Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv
    - Friday-WorkingHours-Morning.pcap_ISCX.csv
    - Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv
    - Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv
```

2. *Import Libraries*: Next, import the necessary libraries from requirements.txt using the following command: `pip install -r requirements.txt`

3. *Run Notebook*: Finally, run the Jupyter Notebook (`project_1_malicious_network_analysis.ipynb`) and execute each cell block sequence from top to bottom.

Note: I have also included the trained model (`pretrained_malicious_traffic_classification_model.pkl`) in the associated zip directory, within a folder titled `Model`, to test the accuracy, as well as the precision of the pre-trained model. To use the pretrained model, line 2 within Cell 9 can be commented, and line 3 can be uncommented.

Dataset

The dataset named CICIDS2017, created by Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani in their paper "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization" comprises network traffic labeled as benign or belonging to various attack types. The traffic was captured over a period of five days, and the attack types are categorized as follows:

- Benign: Non-malicious network traffic
- Botnet: Traffic from compromised devices
- Brute Force: Password guessing attacks (FTP-Patator, SSH-Patator)
- Denial of Service (DoS): Attacks aiming to overload resources (DoS GoldenEye, DoS Hulk, DoS Slowhttptest, DoS Slowloris)
- Distributed Denial of Service (DDoS): DoS attacks using multiple devices
- Port Scan: Network scanning attacks
- Web Attack: Attacks targeting web applications (Bruteforce, XSS, SQL Injection)

To ensure sufficient data for training and evaluation, sample categories with fewer than 100 instances, including Infiltration (n=36), Web Attack SQL Injection(n=21), and Heartbleed(n=11), were removed. The final dataset consists of approximately 2.8 million samples distributed across the attack categories as follows:

```
Benign: 2,271,320
DoS: 251,712
Port Scan: 158,804
DDoS: 128,025
Brute Force: 13,832
Web Attack: 2,159
Botnet: 1,956
```

Data Preprocessing and Splitting

The raw CSV files were concatenated vertically and cleaned to remove non-numeric columns and fix label spacing. The samples were then grouped into broader categories, as listed previously. Any missing values and invariant columns were subsequently dropped from the dataset. The preprocessed data is saved as `preprocessed_network_dataset.csv` for future use or loaded if the file exists.

The labels were integer-encoded, and the data was divided into X (data without labels) and Y (only labels). X was normalized using the MinMaxScaler from the `sklearn` library. X and Y were further split into X_train, X_test, Y_train, and Y_test, with a default ratio of 70% for training data and 30% for test data. However, no validation split was included.

The shape of the four datasets as well as the features and classes are as follows:

```
[+]     X_train shape:  (1979465, 70)
[+]     Y_train shape:  (1979465,)
[+]     X_test shape:  (848343, 70)
[+]     Y_test shape:  (848343,)
[+]     Features:  70
[+]     Classes:  7
```

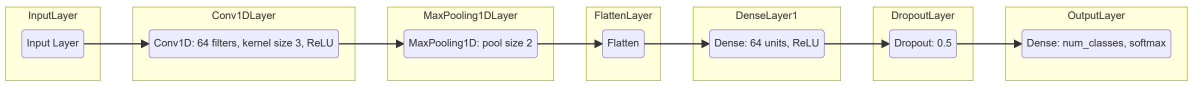
Finally, Y_train and Y_test were one-hot encoded into a format similar to the training data, suitable for training and testing.

Model Architecture

Based on the sequence of traffic data, I implemented a convolutional neural network to classify the traffic patterns. Through testing various hyperparameters, including the number and type of layers, activation functions, and optimizers, an optimal model architecture was identified.

The final model consisted of five layers. The first was a one-dimensional convolutional layer with 64 filters, a kernel size 3, and ReLU activation. This layer extracted spatial features from the input sequences. Next, a max pooling layer with a pool size of 2 reduced dimensionality and performed feature selection. The third layer flattened the feature maps into a one-dimensional vector. The fourth layer was a dense layer with 64 units and ReLU activation. A dropout layer with a rate of 0.5 was also included for regularization. Finally, a softmax output layer performed multi-class classification.

The model was trained for 15 epochs with a batch size of 50, which took approximately 40 minutes on a typical GPU. Through systematic testing of hyperparameters, an optimized 1D convolutional neural network architecture was developed for effective traffic pattern classification. The model architecture is visually represented as follows:

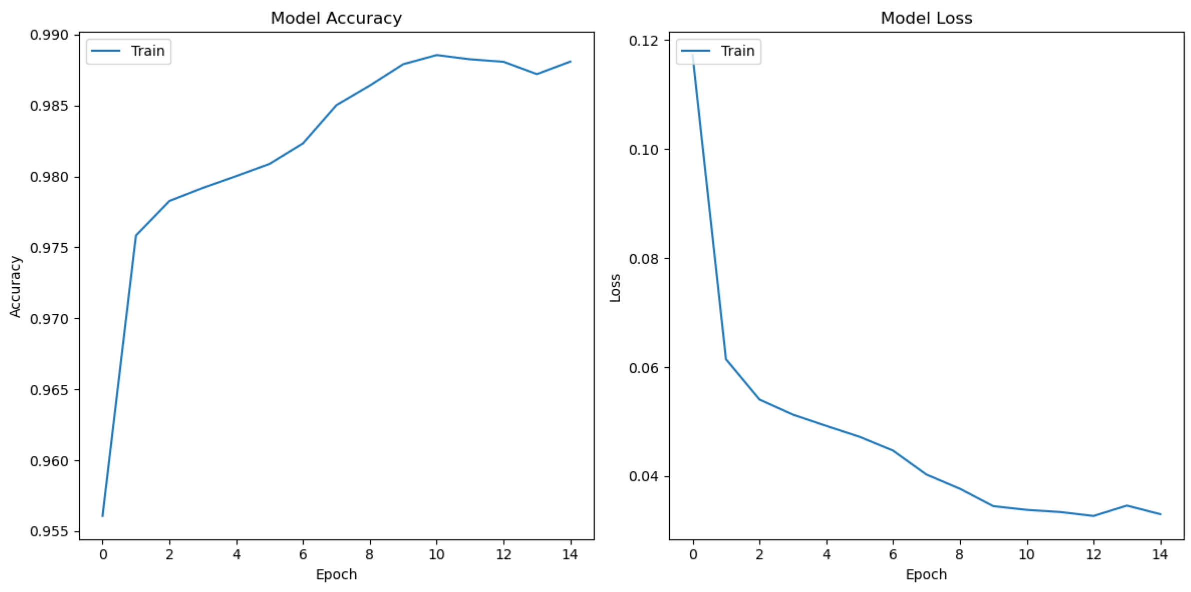


The following was the training and testing output of the model:

```
[+] Training Model...
Epoch 1/15
39590/39590 [==============================] - 155s 4ms/step - loss: 0.1173 -
accuracy: 0.9561
Epoch 2/15
39590/39590 [==============================] - 152s 4ms/step - loss: 0.0615 -
accuracy: 0.9758
Epoch 3/15
39590/39590 [==============================] - 151s 4ms/step - loss: 0.0541 -
accuracy: 0.9783
Epoch 4/15
39590/39590 [==============================] - 151s 4ms/step - loss: 0.0513 -
accuracy: 0.9792
Epoch 5/15
39590/39590 [==============================] - 155s 4ms/step - loss: 0.0492 -
accuracy: 0.9800
Epoch 6/15
39590/39590 [==============================] - 150s 4ms/step - loss: 0.0472 -
accuracy: 0.9809
Epoch 7/15
39590/39590 [==============================] - 148s 4ms/step - loss: 0.0447 -
accuracy: 0.9823
Epoch 8/15
39590/39590 [==============================] - 152s 4ms/step - loss: 0.0403 -
accuracy: 0.9850
Epoch 9/15
```

```
39590/39590 [==============================] - 152s 4ms/step - loss: 0.0377 -
accuracy: 0.9864
Epoch 10/15
39590/39590 [==============================] - 160s 4ms/step - loss: 0.0345 -
accuracy: 0.9879
Epoch 11/15
39590/39590 [==============================] - 155s 4ms/step - loss: 0.0338 -
accuracy: 0.9885
Epoch 12/15
39590/39590 [==============================] - 165s 4ms/step - loss: 0.0334 -
accuracy: 0.9883
Epoch 13/15
39590/39590 [==============================] - 160s 4ms/step - loss: 0.0327 -
accuracy: 0.9881
Epoch 14/15
39590/39590 [==============================] - 161s 4ms/step - loss: 0.0346 -
accuracy: 0.9872
Epoch 15/15
39590/39590 [==============================] - 163s 4ms/step - loss: 0.0330 -
accuracy: 0.9881
[+] Completed training model.
[+] Saving trained model...
[+] Completed saving model.
[+] Testing model accuracy...
26511/26511 [==============================] - 49s 2ms/step - loss: 0.0205 -
accuracy: 0.9946
[+]     Model Accuracy:  0.9946295022964478
```

The training for model accuracy and loss are represented as follows:



Results and Performance

My convolutional neural network model achieves 99.46% accuracy on the test set, surpassing the results reported for ID3 in the paper. My model also demonstrates high precision (0.994), recall (0.994), and F1 score (0.993). While my model requires 2379 seconds of training over 15 epochs, it significantly outperforms the other machine learning algorithms examined in the paper. Specifically, ID3 had the highest weighted average across precision (0.98), recall (0.98), and F1 score (0.98), according to the paper. However, my model achieves even higher scores,

demonstrating its ability to classify benign and malicious traffic accurately. The paper does not specify if ID3's performance applies to identifying malicious vs. benign traffic or individual attack categories, which my model excels at.
Although my model requires longer training time than the algorithms in the paper, the improvement in accuracy warrants this tradeoff. Furthermore, my model surpasses some of the paper's top performers after only 755 seconds and five epochs. It is faster and more accurate than algorithms like KNN and Adaboost. For an IDS task that does not require frequent retraining, my model's superior performance overcomes its longer training requirements.
In summary, my convolutional neural network significantly outperforms the machine learning algorithms examined in the paper despite requiring more extended training, validating its effectiveness for network traffic classification.

References

[1] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, 2018.