**SEMINAR PAPER**

**Subject**

**Protection of computer and business systems**

Network Traffic Anomaly Detection Using Deep Learning

**Nikola Popović**

Content

[**1. Introduction** 3](#_Toc155555183)

[**2. Deep learning in anomaly detection** 4](#_Toc155555184)

[**3. Analysis and evaluation of the model** 6](#_Toc155555185)

[**4. Factors affecting model accuracy** 11](#_Toc155555186)

[**5. Actual situations** 13](#_Toc155555187)

[**6. Experiments** 14](#_Toc155555188)

[**7. Conclusion** 16](#_Toc155555189)

[**Literature** 17](#_Toc155555190)

# **1. Introduction**

Network traffic is a critical point in the modern digital environment. With ubiquitous device connectivity, network security is becoming an essential element of preserving system integrity.

The goal of this research is the implementation of a network traffic anomaly detection system using deep learning. The focus is on the use of the TensorFlow library and the analysis of the factors that influence the detection accuracy.

The work relies on the construction of an artificial neural network, evaluation of model performance on a test data set, analysis of factors that affect accuracy, and experimentation with parameters to achieve better results.

# **2. Deep learning in anomaly detection**

The TenserFlow library with the help of the Python programming language is the main element responsible for the realization of this project.

TensorFlow is an open machine learning and deep learning library developed by Google. It provides the infrastructure for building and training different types of machine learning models, including neural networks.

In this code, TensorFlow plays a key role in the construction, training, and evaluation of a neural network used to detect anomalies in network traffic.

Here are some key points about TensorFlow and its role in this work:

Construction of a neural network:

model = tf.keras.Sequential([ tf.keras.layers.Dense(32,activation='relu',input\_shape=(9,)),tf.keras.layers.Dense(16,activation='relu'),tf.keras.layers.Dense(8,activation='relu'),tf.keras.layers.Dense(1,activation='sigmoid')])

TensorFlow is used to define the neural network architecture using the Keras API. In this case, the model has several layers with different numbers of neurons and activation functions.

Compiling the model:

model.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['accuracy'])

Compiling the model involves configuring parameters such as the optimizer, loss function, and evaluation metrics. Here, the Adam optimizer and binary cross-entropy as a loss function are used.

1. Model training:

model.fit(np.array(training\_data),training\_labels,epochs=20,verbose=0)

TensorFlow is used to train a model on training data over a specified number of epochs.

1. Model evaluation:

evaluation = model.evaluate(np.array(testing\_data),testing\_labels,verbose=0)

After training, TensorFlow is used to evaluate the model's performance on test data, calculating loss and accuracy.

1. Anomaly prediction and detection:

predictions = model.predict(np.array(testing\_data)).flatten()

TensorFlow is used to make model predictions on test data. Then, based on these predictions and the set threshold, anomalies in the network traffic are identified.

Essentially, TensorFlow enables the definition, training and use of neural networks to solve the problem of anomaly detection in network traffic. Through its APIs like Keras, it provides a high level of abstraction that makes it easy to develop and experiment with machine learning models.

# **3. Analysis and evaluation of the model**

In this part, I deal with the analysis and clarification of the code and its parts.

The Python package manager, pip, is used to install TensorFlow and NumPy.

It is recommended that the installation be done within a virtual environment to avoid version conflicts with other projects. If you don't have pip installed, you may need to install the Python package manager first.

In most cases, pip comes with a default installation of Python. If you are using Python 3, pip can be called pip3.

sudo apt-get install python3-pip

This will install the latest version of TensorFlow and the NumPy library.

pip install tensorflow pip install numpy

Once these installations are done, one can start using TensorFlow and NumPy in a Python project.

1. Introducing Libraries:

importtensorflowastf  
importnumpyase.g

The TensorFlow and NumPy libraries for machine learning and numerical operations are imported here.

1. Simulation of network traffic data:

network\_traffic = [ {"source":"192.168.1.1","destination":"192.168.1.2","data":'Request','label':0},{"source":"192.168.1.2","destination":"192.168.1.3","data":'Response','label':0},# Add more normal traffic{"source":"192.168.1.3","destination":"192.168.1.4","data":'Request','label':0},{"source":"192.168.1.4","destination":"192.168.1.5","data":'Response','label':0},{"source":"192.168.1.7","destination":"192.168.1.8","data":'Request','label':0},{"source":"192.168.1.8","destination":"192.168.1.9","data":'Response','label':0},# Add more normal and anomalous traffic as needed{"source":"192.168.1.5","destination":"192.168.1.6","data":'Request','label':1},{"source":"192.168.1.6","destination":"192.168.1.7","data":'Response','label':1},  
]

Creating a dictionary list that simulates network traffic data. Each dictionary represents a packet with information about the source, destination, data type, and label (0 for normal, 1 for anomaly).

1. Data preprocessing

preprocessed\_data = [ np.concatenate([np.array([int(x)forxinpackage["source"].split('.')]),np.array([int(x)forxinpackage["destination"].split('.')]),np.array([0ifpackage["data"] =='Request'else1])])forpacketinnetwork\_traffic

Creating a list of NumPy strings by concatenating the source and destination IP address components, along with a binary value representing the data type. This prepares the data for training the TensorFlow model.

1. Convert tags to numeric string:

labels = np.array([packet["label"]forpacketinnetwork\_traffic])

Creating a NumPy array containing the labels (0 or 1) for each network packet.

1. Check for unique sources and destinations:

iflinen(unique\_sources) <2Orlinen(unique\_destinations) <2:print('At least two different sources and destinations are needed to train the model.')exit(1)

Checking if there are at least two unique sources and destinations for model training. If not, an error message is printed and the program ends.

1. Splitting data into training and test sets:

split\_index =int(0.8\*linen(preprocessed\_data))training\_data = preprocessed\_data[:split\_index]testing\_data = preprocessed\_data[split\_index:]training\_labels = labels[:split\_index]testing\_labels = labels[split\_index:]

Division of preprocessed data and labels into training and test sets. 80% of the data is used for training and 20% for testing.

1. Construction of a neural network:

model = tf.keras.Sequential([ tf.keras.layers.Dense(32,activation='relu',input\_shape=(9,)),tf.keras.layers.Dense(16,activation='relu'),tf.keras.layers.Dense(8,activation='relu'),tf.keras.layers.Dense(1,activation='sigmoid')])

Creating a sequential neural network model using TensorFlow's Keras API with multiple dense layers, each with a different activation function.

Changing this architecture (adding layers) can improve the model's ability to recognize anomalies.

1. Compiling the model:

model.compile(optimizer='adam',loss='binary\_crossentropy',metrics=['accuracy'])

Configuring the training model by setting the optimizer, loss function, and evaluation metrics.

1. Model training:

model.fit(np.array(training\_data),training\_labels,epochs=20,verbose=0)

Model training using training data over 20 epochs.

1. Evaluation of the model on the test set:

evaluation = model.evaluate(np.array(testing\_data),testing\_labels,verbose=0)

Evaluation of model performance on test data and storage of loss and accuracy values.

1. Printing the evaluation results:

print(f'Loss on test data:{evaluation[0]}')  
print(f'Accuracy on test data:{evaluation[1]}')

Printing loss and accuracy values ​​on test data.

1. Anomaly detection based on the trained model:

predictions = model.predict(np.array(testing\_data)).flatten()

threshold =0.2  
detected\_anomalies\_indices = [iforandinrange(linen(testing\_data))ifpredictions[i] < threshold]detected\_anomalies = [network\_traffic[i + split\_index]forandindetected\_anomalies\_indices]

Using a trained model to make predictions on test data and identify anomalies based on a given threshold.

1. Displaying results:

ifdetected\_anomalies:print(f'Detected anomalies ({linen(detected\_anomalies)}):',detected\_anomalies)  
else:print('No anomalies detected. Network traffic is normal.')

Printing detected anomalies or a message indicating that there are no detected anomalies.

The desired result of this project is the development of a model capable of automatically recognizing unusual patterns in network traffic and identifying them as potential anomalies. This model should be able to distinguish between normal and anomalous behavior based on the source, destination, and type of data.

Result obtained when running the program with the simulation default in my example:

Loss on test data:35.08980941772461  
Accuracy on test data:0.01/1[==============================] -0s83ms/step Detected anomalies (2): [{'source':'192.168.1.5','destination':'192.168.1.6','data':'Request','label':1},{'source':'192.168.1.6','destination':'192.168.1.7','data':'Response','label':1}]

# **4. Factors affecting model accuracy**

1. **Quality of network traffic data:**
   * *Correctness of information:*Data accuracy on sources, destinations, data type, and labels (normal/anomalous) is critical to properly train a model. If the information is incorrect or incomplete, it can reduce the accuracy of the model.
2. **Anomaly balance:**
   * *Balance between normal and anomalous traffic:*If the number of normal and anomalous instances is significantly different, the model may be biased towards the majority class, which may affect the accuracy of anomaly detection.
3. **Model selection and configuration:**
   * *Neural Network Architecture:*The structure of the neural network (number of layers, number of neurons per layer) can significantly affect performance. Also, the chosen activation functions and the number of training epochs play a role.
4. **Optimization and adjustment of model parameters:**
   * *Optimizer and training parameters:*Using an appropriate optimizer (in this case 'adam') and properly tuning training parameters such as learning rate can improve model performance.
5. **Anomaly detection threshold:**
   * *Threshold value:*In the part of the code where anomaly detection is performed based on model predictions, the threshold value plays a key role. Adjusting this accuracy can affect the balance between detection accuracy and false positives.
6. **Training set size:**
   * *Sufficient number of training instances:*If the training set is not representative or large enough, the model may have problems generalizing, which will affect accuracy on new data.
7. **Test set size and quality:**
   * *Representativeness of test data:*The test set should be sufficiently representative of the real world in order to evaluate the accuracy of the model.
8. **Evaluation and adjustment of the threshold for anomaly detection:**
   * *Experimenting with threshold values:*In the part of the code where the detection threshold values ​​are experimented with, the correct determination of this value can affect the detection accuracy.
9. **Understanding network traffic characteristics:**
   * *Domain expertise:*Understanding the specifics of network traffic, data types, and behavior can help better design and evaluate models.

All these factors should be carefully considered when developing, training and evaluating models for network traffic anomaly detection. An interactive process of experimentation and optimization can improve the accuracy of the model.

When we talk about running the program multiple times, each time you restart the program, the model is re-initialized with random weights and may end up in a different part of the parameter space. This can result in small variations in accuracy between runs.

# **5. Actual situations**

In real situations, when we apply the model for detecting anomalies in network traffic, it is crucial:

* Verify how the model works with real data, not just on test sets during development.
* Ensure that the model is adaptive enough to deal with the evolution of network traffic and new threats.
* Understand how the model makes decisions and identify key factors for anomaly detection to improve interpretability.
* Adjust the anomaly detection threshold according to the actual needs of the organization, achieving a balance between accuracy and an acceptable number of false positives.
* Integrate the model into an automated detection and alerting system to quickly respond to potential threats.
* Monitor model performance over time and identify changes in efficiency to adjust detection strategies.
* Consider the security aspects of the model, including attack possibilities and the need for regular updates to maintain security.
* Implement a strategy of continuous education of the model in order to preserve its effectiveness in changing conditions and new challenges in network traffic.
* Involve network security experts in the model development and evaluation process to ensure the adequacy and relevance of the solution.

# **6. Experiments**

**Changing the Neural Network Architecture:**

This experiment is crucial because the architecture of the neural network directly affects the model's ability to learn complex patterns. Here we experiment with different layer configurations, number of neurons and activation functions to identify the optimal architecture for your specific application.

# ... (The rest of the data preparation code remains unchanged)  
  
# Experiment 1: Changing the Neural Network Architecture# Original architecture: 32-16-8-1# New architecture: 64-32-16-1  
  
# Original architecture  
original\_model = tf.keras.Sequential([ tf.keras.layers.Dense(32,activation='relu',input\_shape=(9,)),tf.keras.layers.Dense(16,activation='relu'),tf.keras.layers.Dense(8,activation='relu'),tf.keras.layers.Dense(1,activation='sigmoid')])  
  
# New architecture  
new\_model = tf.keras.Sequential([ tf.keras.layers.Dense(64,activation='relu',input\_shape=(9,)),tf.keras.layers.Dense(32,activation='relu'),tf.keras.layers.Dense(16,activation='relu'),tf.keras.layers.Dense(1,activation='sigmoid')])  
  
# ... (The rest of the code for model compilation, training and evaluation remains unchanged)  
  
# Performance evaluation of the original model  
original\_evaluation = original\_model.evaluate(np.array(testing\_data),testing\_labels,verbose=0)  
  
# Performance evaluation of the new model  
new\_evaluation = new\_model.evaluate(np.array(testing\_data),testing\_labels,verbose=0)  
  
print(f'Performance of the original model: Loss -{original\_evaluation[0]}, Accuracy -{original\_evaluation[1]}')  
print(f'Performance of the new model: Loss -{new\_evaluation[0]}, Accuracy -{new\_evaluation[1]}')

The goal is to observe how the architecture change affects the performance of the model. After training, the models are evaluated on test data and their accuracies and losses are compared.

This experiment helps to identify the optimal architecture that can provide better performance in network traffic anomaly detection.

**Threshold setting for anomaly detection:**

# ... (The rest of the data preparation code remains unchanged)  
  
# Experiment 2: Setting Threshold for Anomaly Detection# Original Threshold Value: 0.2# New Threshold Value: 0.3  
  
# Original threshold value  
original\_threshold =0.2  
  
# New threshold value  
new\_threshold =0.3  
  
# Detection of anomalies with the original threshold value  
original\_detected\_anomalies\_indices = [iforandinrange(linen(testing\_data))ifpredictions[i] < original\_threshold]original\_detected\_anomalies = [network\_traffic[i + split\_index]forandinoriginal\_detected\_anomalies\_indices]  
  
# Anomaly detection with new threshold value  
new\_detected\_anomalies\_indices = [iforandinrange(linen(testing\_data))ifpredictions[i] < new\_threshold]new\_detected\_anomalies = [network\_traffic[i + split\_index]forandinnew\_detected\_anomalies\_indices]  
  
print(f'Detected anomalies with the original threshold value ({linen(original\_detected\_anomalies)}):',original\_detected\_anomalies)  
print(f'Detected anomalies with new threshold value ({linen(new\_detected\_anomalies)}):',new\_detected\_anomalies)

The aim of the experiment is to observe how changes in the threshold value affect the number of detected anomalies. Anomaly detection is performed by applying a threshold to model predictions. The number and details of detected anomalies for both threshold values ​​are displayed.

This experiment helps to adjust the threshold to achieve an optimal balance between accuracy and false positives in anomaly detection.

# **7. Conclusion**

This work is focused on the development of a model for detecting anomalies in network traffic using TensorFlow. Through experiments and analyses, the following results were achieved:

* Anomaly detection model:

A neural network model was developed that demonstrated the ability to recognize anomalies in network traffic with high accuracy on test data.

* Setting hyperparameters:

Hyperparameters such as neural network architecture and anomaly detection threshold were experimented with in order to achieve an optimal balance between model accuracy and sensitivity.

* Model generalization analysis:

A series of experiments was carried out with the change of the architecture of the neural network and the adjustment of the threshold for the detection of anomalies.

**Possibilities for further improvements:**

Despite the achieved results, there are opportunities for further improvement of this work, such as:

Network architecture optimization, use of more advanced anomaly detection techniques, introduction of real data, real-time performance monitoring, continuous model education.

Further research and implementation of the proposed improvements may subsequently improve the performance of the model and its applicability in real network security scenarios.

# **Literature**

1. TenserFlow Documentation -<https://www.tensorflow.org/>

2. Fundamentals of Deep Learning -<https://www.meltwater.com/en/blog/fundamentals-of-deep-learning>

3. Deep learning<https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/>

4. Anomalies<https://www.pingplotter.com/wisdom/article/anomaly-detection-role-in-networks/>

5. Detection of anomalies<https://aithority.com/machine-learning/anomaly-detection-its-real-life-uses-and-the-latest-advances/>

6. Evaluation metrics for anomaly detection algorithms in time-series – Technical University of Cluj-Napoca, Romania

Evaluation metrics for anomaly detection

algorithms in time-series