Anomaly Detection in a Data Center with IoT Sensors

Ayushi Savani
W Booth School of Engineering Practise and Technology
McMaster University
savania@mcmaster.ca

Twinkal Togadiya
W Booth School of Engineering Practise and Technology
McMaster University
togadiyt@mcmaster.ca

Sebastian Rivera
W Booth School of Engineering Practise and Technology
McMaster University
riveraas@mcmaster.ca

Abstract—*The connected world of the 21st century has allowed remote and automated systems in the home, school and workplace. Automated processes in the workplace have impacted how people complete their jobs and how companies invest their resources to save cost and improve their products. In the industrial manufacturing industries, there has been an explosion of remote monitoring on the status of processes, replacing humans and introducing sensors and actuators that are connected via the internet. Vulnerabilities are inevitable in this internet of things communication pathways and Deep learning methods have been reliable in determining when anomalies occur. These anomalies are usually cyber-attacks to interrupt and manipulate the machinery and process in a negative manner for ransom. Other institutions like the government, hospitals and the transportation industry are all using remote monitoring systems to detect these anomalies and prevent the nest outage. Using a DAD dataset different models will be trialed and evaluated.*

I. Introduction

Through machine different machine learning methodologies, the execution of the analysis and improvement of the current available models to detect anomalies in a DAD dataset containing anomaly traffic. In this modern age of connected devices, various industries are implementing cyber security in all their monitoring and autonomous systems that can be compromised by cyber-attacks. Using a DAD (Dataset for Anomaly Detection) various machine learning models like RNNs, LTSM and FNN models will be implemented to detect the anomalies in the dataset. As more industries adopt connected networks of sensors and actuators the need for a model to detect anomalies live is necessary to maintain the security and systems running smoothly. In this report the models mentioned above will be explored thoroughly, by experimentation.

II. Problem Statement

The IoT data analysis in this report will shed light on the improvement of remote monitoring, through three types of anomalies being duplication, interception and modification; the various DAD files must be examined to carry out appropriate research through the implementation of the various models mentioned above. The lack of analysis on any traffic behavior is concerning and models to detect anomalies must be implemented to take proactive or reactive action based on the attack. This analysis will be advantageous in providing proactive and reactive measures to maintain the system operating under cyber-attacks or systems faults. Preprocessing.

III. Review / Background

Models such as the ARIMA AutoRegressivce Integrated Moving Average) is the most common model for forecasting time series data[1]. There are three main segments of the model that make it work effectively but during high computational resources. Autocorrelation is when the model searches for how the dataset relates to previous data in its history. This search is conducted over time periods referred to as 'lags' how many lags can be attributed to how many sets of data are in the previous time step. Stationary time series, forcing the time series to be stationary is achieved by calculating the changes between consecutive observations, this is done for a consistent mean and variance over time. These processes and methods of the ARIMA are usually plotted on using 'Auto ARIMA too in R' this is a tool in R programming that will dictate the best ARIMA model, the precision and agility of the calculation is also improved [1].

Many methods have been proposed and anomaly detection has been the subject of in-depth research. Using the idea of isolation, the effective algorithm Isolation Forest finds anomalies in high-dimensional datasets. Neural networks called autoencoders can recognize typical data patterns and identify deviations as anomalies. When it comes to time-series data, LSTM networks are efficient, capturing temporal dependencies to identify anomalies. This project will explore an LSTM (Long Short-Term Memory) network, this RNN (recurrent neural network) will also combine multiple other models that will execute majority voting to ensure the network can predict anomalies effectively and accurately.

IV. Theory and Datasets

To implement the most effective solution in the RNN, the following models were combined for their veracious advantages and improvements to the overall prediction of the dataset.

Random forest classifier

This is a machine learning model included in the 'scikit-learn library' and primary used in classification objectives. It is a "meta estimator that fits the number of decision tree classifiers on various sub-samples of the dataset" [3]. To improve predictive accuracy and overfitting this classifier uses averaging.

Isolation Forest

This model is used to split a random value of a selected feature between the minimum and maximum, the goal is to perform outlier detection in “high-dimensional datasets”, this DAD contains 101,583 packs with three main MQTT messages, making the inclusion of this method necessary to accurately predict [1][4].

LSTM (Long-Short Term Memory)

This model is very effective for time series datasets as it excels in sequence tasks capturing and using the long-term dependencies in the data. There is a cell state that it uses to store information about past cell inputs [5].

The dataset used in this project is called DAD (Dataset for Anomaly Detection), and it was developed specifically for evaluating machine learning models in the context of anomaly detection. This dataset offers a comprehensive framework for analyzing the performance of different models because it contains a wide range of normal and anomalous data points.

The DAD contains 101,583 packets where 63.3% are MQTT packets and 16% of these packets are anomalies [1].

Each packet contains the following characteristics according to:

- **frame.number**: frame number in the PCAP package.
- **ip.src**: MQTT client IP.
- **tcp.srcport**: MQTT client TCP source port.
- **mqtt.clientid**: MQTT client identifier.
- **mqtt.msgid**: Token identifier or message number.
- **label**: 0 indicates that it is a packet without anomalies and 1 that the packet is part of a flow that has been altered.

For seven days the following occurred:

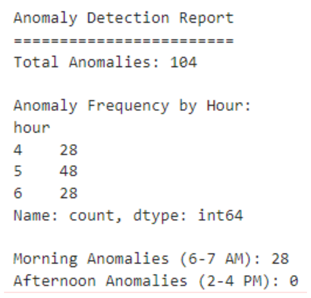
- Monday 21st: there are no anomaly packets.
- Tuesday 22nd: some packets have been removed, so packets are not labeled as anomaly.
- Wednesday 23rd: a modification of packets is made between 4 and 6 h.
- Thursday 24th: insertion of packets in less than 5 min at 3 h.
- Friday 25th: a mix of interception, duplication and modification is done at 6 h and between the 14–16h.
- Saturday 26th: a mix of interception, duplication and modification is done at 6 h and between the 14–16h.
- Sunday 27th: there are no anomaly packets.

IV. Implementation Details

The anomaly detection models are implemented through several crucial phases, including data preprocessing and model evaluation. Development and improvement of machine learning models capable of efficiently identifying anomalies in the DAD (Dataset for Anomaly Detection) though combining three models with majority voting.

A. Data Preprocessing and Visualization

The following report and bar graph is representing the Wednesday where 104 anomalies were detected between the hours of 4 and 6 pm.



```
Anomaly Detection Report
========================
Total Anomalies: 104

Anomaly Frequency by Hour:
hour
4    28
5    48
6    28
Name: count, dtype: int64

Morning Anomalies (6-7 AM): 28
Afternoon Anomalies (2-4 PM): 0
```

Figure 1. Wed 23rd Anomaly Report

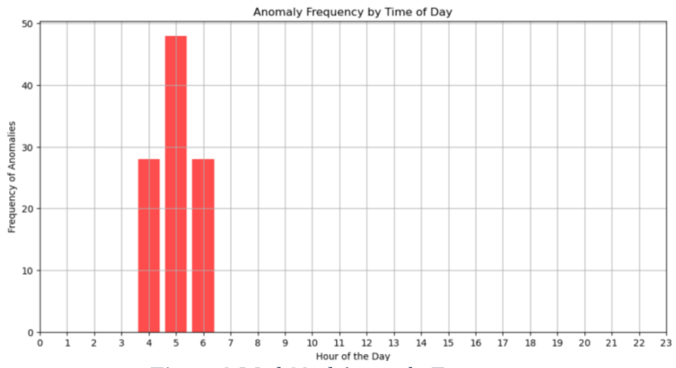


Figure 2.Wed 23rd Anomaly Frequency

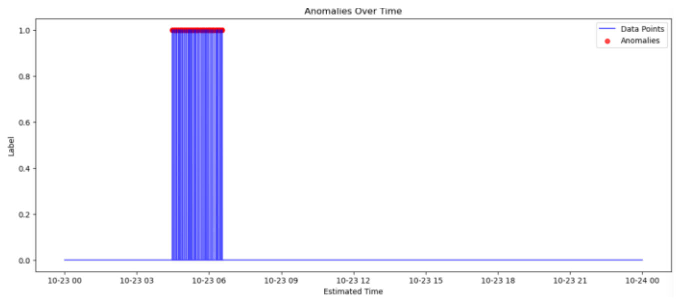


Figure 3. Wed 23rd Anomalies

To visualize and produce an anomaly report for each day the time needed to be estimated based on the ‘frame.number’ in each csv file, please refer to “SEP 769 Preprocessing Data Final Project.py”. This is confirmed by the

DAD data set characteristics outlined earlier where Wednesday experienced the anomalies in those times. To classify the anomalies the code assumes that a value of 0 is normal and other values are anomalies.

anomalies = data[data['label'] != 0]

For the data to be processed into a usable dataset for deep learning models to use and other functions to visualize a hot encoder is used specifically the ‘ip.src’ column, each different ip.src is treated as a category and different ips are treated accordingly. Each ip address is converted into a binary vector[6]. While other variable like ‘tcp.srcport’, ‘mqtt.clientid’, and ‘mqtt.msgid’ are normalized to ensure each variable is on the same level of magnitude to influence the learning process[7].

C. Model Training and Tuning

To use the DAD to implement the proposed methodology discussed above the data needed to be concatenated together for analysis. To handle any missing values in the data a place holder of 0 was used. The dataset is split through means of the scikit-learn library into the following proportions:

Training set: 60%

Validation Set: 20%

Test Set: 20%

The weights are then calculated: Class weights:

{0: 0.5026224208738104, 1: 95.83176100628931}

A total of 100 Epoch were ran to ensure a well-trained model but because of the implementation of the early stopping callback function, a patience of 3 was used to prevent over fitting. A total of 9 epochs were necessary to produce the results.

V. Explanation of the Source Code

A. Data Processing

The following will explain noteworthy code segments from the SEP_769_Final_Project_Code_FINAL_Group7.ipynb. The data contained missing information therefore the missing values were replaced by zeros. To ensure there no issues are encountered during model training.

```
   data['mqtt.clientid'] =
data['mqtt.clientid'].fillna(0)
    data['mqtt.msgid'] =
data['mqtt.msgid'].fillna(0)
```

The categorical data needs to be encoded to be able to be used, where the ip.src are converted to strings.

```
data['ip.src'] =
data['ip.src'].astype(str)
    encoder = OneHotEncoder()
    ip_encoded =
encoder.fit_transform(data[['ip.src']]).t
oarray()
    ip_encoded_df =
pd.DataFrame(ip_encoded,
columns=encoder.get_feature_names_out(['i
p.src']))
```

The numerical features are normalized by the StandardScaler functions for all of them to have a S.D of 1 and mean of 0.

```
 scaler = StandardScaler()
    data[['tcp.srcport', 'mqtt.clientid',
'mqtt.msgid']] =
scaler.fit_transform(data[['tcp.srcport',
'mqtt.clientid', 'mqtt.msgid']])
```

The feature and labels are processed are combined according to scaled and encoded features.

```
 features =
pd.concat([data.drop(['ip.src', 'label'],
axis=1), ip_encoded_df], axis=1)
    return features, data['label']
```

The DAD is split into 60% training, 20% validation and 20% Testing.

```
X_train, X_temp, y_train, y_temp =
train_test_split(X, y, test_size=0.4,
random_state=42)
X_val, X_test, y_val, y_test =
train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)
```

The weights are calculated.

```
class_weights =
compute_class_weight('balanced',
classes=np.unique(y_train), y=y_train)
class_weights_dict =
dict(enumerate(class_weights))
```

SMOTE is implemented to oversample the minority class.

```
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote =
smote.fit_resample(X_train, y_train)
```

Training the Random Forest Model using the calculated weights.

```
rf_model =
RandomForestClassifier(class_weight=class
_weights_dict, random_state=42)
rf_model.fit(X_train_smote,
y_train_smote)
```

Training the Isolation Forest Model with the SMOTE balance dataset outputs.

```
iso_forest =
IsolationForest(contamination=0.1,
random_state=42)
iso_forest.fit(X_train_smote)
```

LSTM with dropout layers to prevent overfitting

```
lstm_model = Sequential([
    LSTM(50,
input_shape=(X_train_reshaped.shape[1],
1), return_sequences=True),
    Dropout(0.2),
    LSTM(50),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
```

Training the LSTM with the calculated weights. With 100 epochs and early stopping function.

```
history_lstm =
lstm_model.fit(X_train_reshaped,
y_train_smote, epochs=100, batch_size=64,
validation_data=(X_val_reshaped, y_val),
verbose=1,
class_weight=class_weights_dict,
callbacks=[early_stopping])
```

This makes predictions using the LSTM models and majority voting based on the three models.

```
y_pred_train_lstm =
(lstm_model.predict(X_train_reshaped) >
0.5).astype(int)

y_pred_train_combined = (y_pred_train_iso
+ y_pred_train_lstm.flatten() +
rf_model.predict(X_train_smote)) > 1
```

```
y_pred_train_combined = (y_pred_train_iso
+ y_pred_train_lstm.flatten() +
rf_model.predict(X_train_smote)) > 1
y_pred_val_combined = (y_pred_val_iso +
y_pred_val_lstm.flatten() +
rf_model.predict(X_val)) > 1
y_pred_test_combined = (y_pred_test_iso +
y_pred_test_lstm.flatten() +
rf_model.predict(X_test)) > 1
```

The model results are printed

```
print("Combined Model Performance")
print(f"Train ROC-AUC:
{roc_auc_score(y_train_smote,
y_pred_train_combined)}")
print(f"Val ROC-AUC:
{roc_auc_score(y_val,
y_pred_val_combined)}")
print(f"Test ROC-AUC:
{roc_auc_score(y_test,
y_pred_test_combined)}")
print(f"Test F1-score: {f1_score(y_test,
y_pred_test_combined)}")
print(f"Test Precision:
{precision_score(y_test,
y_pred_test_combined)}")
print(f"Test Recall:
{recall_score(y_test,
y_pred_test_combined)}")
```

This plots the training history

```
plt.plot(history_lstm.history['loss'],
label='Training Loss')
plt.plot(history_lstm.history['val_loss']
, label='Validation Loss')
```

```
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

The following code plots the ROC curves and confusion matrices

```
def plot_roc_curve(y_true, y_pred,
title='ROC Curve'):
    fpr, tpr, _ = roc_curve(y_true,
y_pred)
    plt.plot(fpr, tpr, marker='.')
    plt.plot([0, 1], [0, 1], linestyle='-
-')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.show()

plot_roc_curve(y_train_smote,
y_pred_train_combined, 'ROC Curve for
Training Data')
plot_roc_curve(y_val,
y_pred_val_combined, 'ROC Curve for
Validation Data')
plot_roc_curve(y_test,
y_pred_test_combined, 'ROC Curve for Test
Data')
```

```
def plot_confusion_matrix(y_true, y_pred,
title='Confusion Matrix'):
    cm = confusion_matrix(y_true, y_pred)
    disp =
ConfusionMatrixDisplay(confusion_matrix=c
m)
    disp.plot(cmap=plt.cm.Blues)
    plt.title(title)
    plt.show()

plot_confusion_matrix(y_train_smote,
y_pred_train_combined, 'Confusion Matrix
for Training Data')
plot_confusion_matrix(y_val,
y_pred_val_combined, 'Confusion Matrix
for Validation Data')
plot_confusion_matrix(y_test,
y_pred_test_combined, 'Confusion Matrix
for Test Data')
```

While the library imports, data processing, remains the same for the MLP model notable differences will be shown below along with its specific functions.

The set up of both models remain the same up until the hyper parameters for random forest using GridSearchCV

```
rf_params = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'class_weight':
[class_weights_dict]
}
rf_model =
GridSearchCV(RandomForestClassifier(ran
dom_state=42), rf_params, cv=5,
scoring='f1', n_jobs=-1)
rf_model.fit(X_train_smote,
y_train_smote)
print(f"Best Random Forest Parameters:
{rf_model.best_params_}")
```

n_estimators, are the number of trees in the forest, while a higher number can improve performance it also increases the total computation time.
max_depth, sets the depth of each tree, a lower depth will prevent over fitting but can lead to opposite as well, underfitting.
min_samples_split, will indicate the minimum number of samples needed to split the node.
min_samples_leaf, at the leaf this would set the minimum number needed
class_weight, uses the calculated weight from the previous line in the code

The GridSearchCV function provide a cross validation techniques to search the hyperparameters to find the best model formation based on the F1 score.

The XGBoost and Isolation Forest Model is trained below

```
xgb_model = XGBClassifier(
    scale_pos_weight=class_weights_dict[1
],
    max_depth=6,
    n_estimators=200,
    learning_rate=0.1,
    random_state=42
)
xgb_model.fit(X_train_smote,
y_train_smote)

# Train Isolation Forest model
iso_forest =
IsolationForest(contamination=0.1,
random_state=42)
iso_forest.fit(X_train_smote)
```

scale_pos_weight, balances the negative and positive weights of the imbalanced classes
max_depth, depth of the tree, lower depth prevent overfitting
n_estimators, boosing rounds
learning_rate, step size at each iteration

The FFN model is now

```
fnn_model = Sequential([
    Flatten(input_shape=(X_train_smote.sh
ape[1], 1)),
    Dense(128, activation='relu',
kernel_regularizer=l2(0.01)),
    Dropout(0.8),
    Dense(64, activation='relu',
kernel_regularizer=l2(0.01)),
    Dropout(0.8),
    Dense(32, activation='relu',
kernel_regularizer=l2(0.01)),
    Dropout(0.8),
    Dense(1, activation='sigmoid')
```

This sequential FNN model allows the construction of the model in a forward manner, by adding layers by specifying the order of the layers. The sequential model nature has the following workings within it:
Flatten, this manipulates the data into a 1D array
Dense, the ReLU activation function is activated and will allow non-linearity by converting all the negative values to zeros and the positive values unchanged. This first layer compasses 128 layers, this high number of layers will capture complex pattern in the dataset.

Kernel_regularizer, this applies L2 regularization to keep the model from over fitting penalizing large weights

Each time the Dense function is called a new layer is made with different and less layers like 64 and 32.

Compiling the FNN model

```
optimizer = Adam(learning_rate=0.001)
fnn_model.compile(optimizer=optimizer,
loss='binary_crossentropy',
metrics=['accuracy'])
fnn_model.summary()
```

The optimizer **Adam** is an algothirm that can adapt, it being at 0.001 and then changes depending on the loss gradient during each update[8].
The **binary_crossentropy,** is a loss function, a standard for binary classification that determines the input into one of the two categories[9].

The early stopping callback function was also used in the FNN, as well as 100 with a batch size of 64.

A manual threshold of 0.5 as also set

```
manual_threshold = 0.5
```

This manual threshold was implemented to work in this binary classification, it was 0.5 since it is a balanced decision boundary [10].

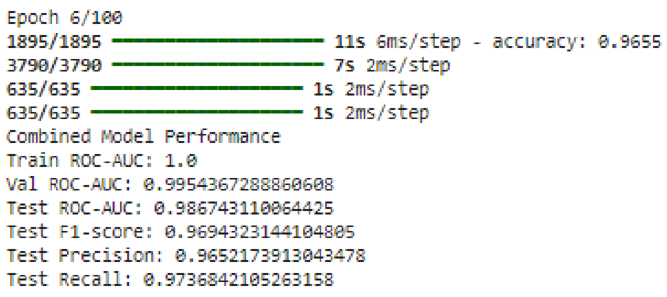
Majority voting was also implemented before the stacking classifier.

```
stacking_model = StackingClassifier(
    estimators=[
        ('rf', rf_model.best_estimator_),
        ('xgb', xgb_model)
    ],
    final_estimator=LogisticRegression(),
    cv=5,
    n_jobs=-1
)
```

This stacking allows the full features of both models to be utilized, the cv = 0.5 is a 5 cross fold cross-validation, this validates the model's performance

Once this stacking segment is complete then the code continues like the RNN to print and plot the results. .

V. Results and Discussion



```
Epoch 6/100
1895/1895 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9655
3790/3790 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step
635/635 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
635/635 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Combined Model Performance
Train ROC-AUC: 1.0
Val ROC-AUC: 0.9954367288860608
Test ROC-AUC: 0.986743110064425
Test F1-score: 0.9694323144104805
Test Precision: 0.9652173913043478
Test Recall: 0.9736842105263158
```

Figure 4. LSTM Results

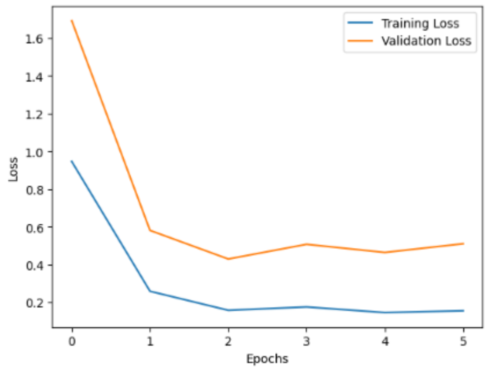


Figure 5. LSTM Epochs Results

The ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) score on the training set is 1.0, which means the model performs perfectly. The model can completely distinguish between anomalies vs. normal instances in the training data without any errors. While the Val ROC-AUC is 0.995 on the validation dataset and 0.986 on an unseen dataset. These scores near 1.0 signify that the model is very accurate and efficient not only on the DAD but future unseen data that can be generated from real world environments.

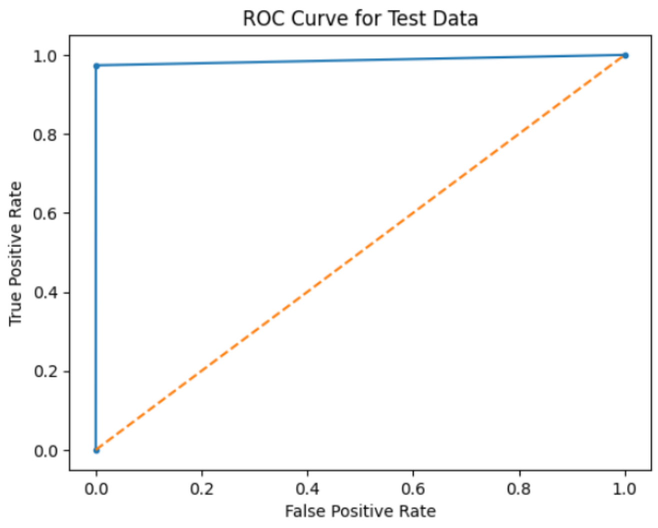


Figure 6. RNN ROC Curve for test data

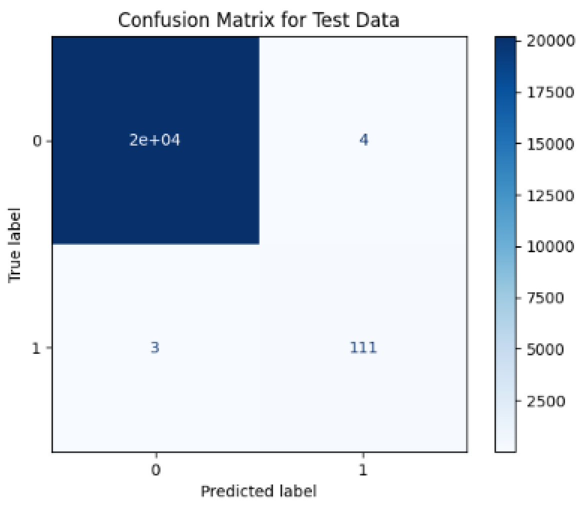
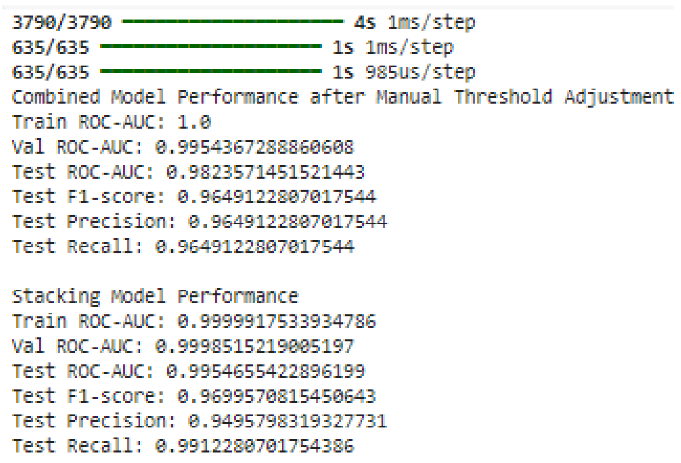


Figure 7.RNN Confusion Matrix

The confusion matrix depicts that the model works well in identifying negative (20k) and positive (111) instances, while also very effective in minimizing errors. The training and validation data for the ROC-AUC are very close to each other, this suggests that if overfitting is occurring it is very minimal and does not mean that the model's performance is being manipulated by these results. Therefore, the model's performance is not only expected to perform this well on this data set but also on unseen data sets.

For the FNN model the results are as follows



```
3790/3790 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step
635/635 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
635/635 ━━━━━━━━━━━━━━━━━━━━ 1s 985us/step
Combined Model Performance after Manual Threshold Adjustment
Train ROC-AUC: 1.0
Val ROC-AUC: 0.9954367288860608
Test ROC-AUC: 0.9823571451521443
Test F1-score: 0.9649122807017544
Test Precision: 0.9649122807017544
Test Recall: 0.9649122807017544

Stacking Model Performance
Train ROC-AUC: 0.9999917533934786
Val ROC-AUC: 0.9998515219005197
Test ROC-AUC: 0.9954655422896199
Test F1-score: 0.9699570815450643
Test Precision: 0.9495798319327731
Test Recall: 0.9912280701754386
```

Figure 8. FNN Sequential Results

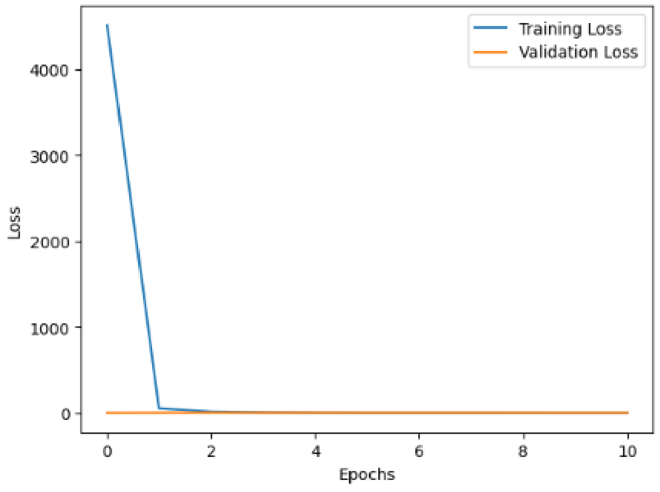


Figure 9. FNN Epochs Results

The ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) stacking score on the training set is 0.999, which means the model performs almost perfectly. The model can completely distinguish between anomalies vs. normal instances in the training data without any errors. While the Val ROC-AUC is 0.999 on the validation dataset and 0.995 on an unseen dataset. These scores near 1.0 signify that the model is very accurate and efficient not only on the DAD but future unseen data that can be generated from real world environments.

The ROC curve for test data demonstrates that the model has a Ture Positive Rate of 1 therefore it correctly identifies positive cases; and a False Positive Rate of 0 therefore it will never incorrectly identify negative cases as positive.

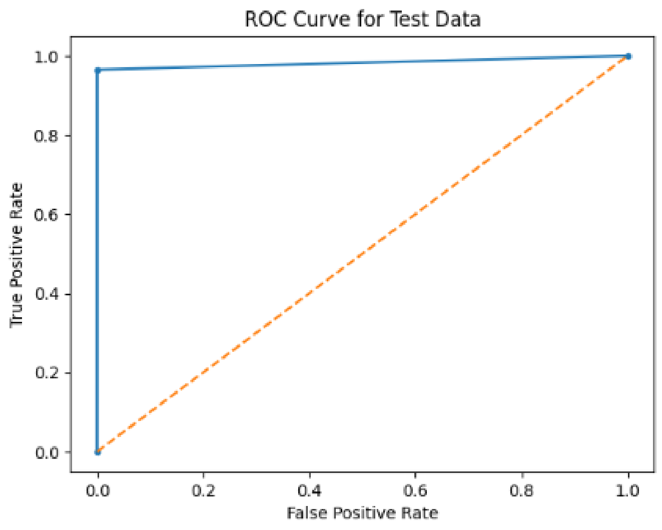


Figure 10. FNN ROC Curve for test data

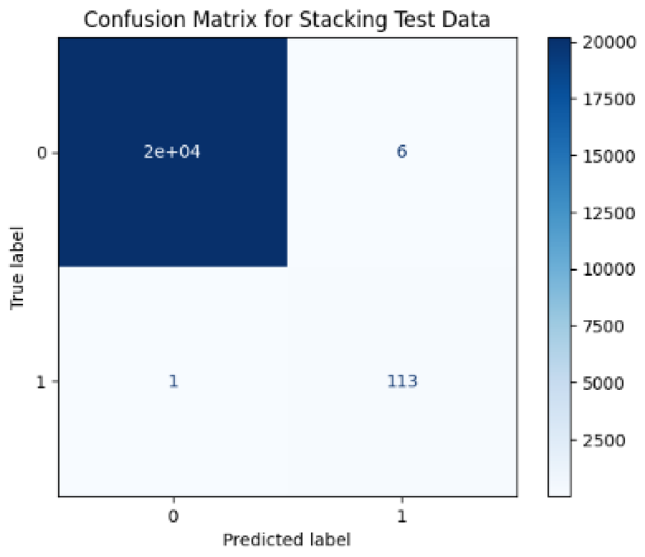


Figure 11.FNN Confusion Matrix

The confusion matrix depicts that the model works well in identifying negative (20k) and positive (113) instances, while also very effective in minimizing errors. The training and validation data for the ROC-AUC are very close to each other, this suggests that if overfitting is occurring it is very minimal and does not mean that the model's performance is being manipulated by these results. Therefore, the model's performance is not only expected to perform this well on this data set but also on unseen data sets.

VI. CONCUSION

The RNN LSTM's and FNN's ability to handle temporal data, combined with majority voting from Random Forest (ensured high accuracy and robustness) and Isolation Forest(ensured the proper identification of outlier and anomalies) models, was successfully implemented and trained in a controlled environment. This approach yielded near-perfect results in accurately identifying anomalies. The effectiveness of the model was significantly enhanced using SMOTE (Synthetic Minority Over-sampling Technique), which played a crucial role in mitigating class imbalance by oversampling the minority class. This ensured that the model did not become biased during training and was able to generalize well, resulting in a highly accurate anomaly detection system.

VII. Recommendations for Future Work

Utilize Larger Datasets: Expand5ing the dataset to encompass a broader array of scenarios and anomalies can enhance the model’s accuracy and generalizability.

Explore Advanced Model Architectures: Investigating and incorporating more advanced or hybrid model architectures can potentially improve performance. For instance, combining autoencoders with attention mechanisms or experimenting with

ensemble methods that leverage the strengths of multiple models might yield better results.

Deploy in Real-Time Systems: Integrating the anomaly detection models into real-time monitoring systems can facilitate timely responses to detected anomalies.

Enhance Robustness and Security: Further research into improving the robustness of models against adversarial attacks and noisy data is essential.

Conduct Cross-Domain Testing: Applying the models to different domains or industries. Testing models in diverse operational environments will help validate their robustness and adaptability to different types of data and anomalies.

References

[1] A. Gil-de-Castro, M. J. Lucena-Molina, J. Pérez-Ortiz, and F. Gómez-Fernández, "ARIMA Models for Reliable Predictive Maintenance Using Data from IoT Sensors," *Sensors*, vol. 20, no. 13, p. 3745, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/13/3745

[2] L. Vigoya, D. Fernandez, V. Carneiro, and F. Cacheda, "Annotated Dataset for Anomaly Detection in a Data Center with IoT Sensors," Sensors, vol. 20, no. 13, p. 3745, 2020. [Online]. Available: https://github.com/dad-repository/dad.

[3] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[4] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011. [Online]. Available: https://scikit-learn.org/stable/modules/outlier_detection.html#isolation-forest.

[5] "Deep Learning | Introduction to Long Short-Term Memory," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/.

[6] "Target encoder," Scikit-learn, 2024. [Online]. Available: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_target_encoder.html.

[7] "Normalization," Scikit-learn, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/preprocessing.html#normalization.

[8] J. Brownlee, "Adam Optimization Algorithm for Deep Learning," Machine Learning Mastery. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[9] J. Brownlee, "Binary Classification Tutorial with the Keras Deep Learning Library," Machine Learning Mastery. Available: https://machinelearningmastery.com/binary-classification-tutorial-with-the-keras-deep-learning-library/.

[10] J. Brownlee, "Threshold Moving for Imbalanced Classification," Machine Learning Mastery. Available: https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/.