Detecting Cyber Attacks in Industrial Control Systems Using Convolutional Neural Networks

Conference Paper · October 2018

DOI: 10.1145/3264888.3264896

Moshe Kravchik
Ben-Gurion University of the Negev

Detecting Cyber Attacks in Industrial Control Systems Using Convolutional Neural Networks

Moshe Kravchik, Asaf Shabtai
Department Of Software and Information Systems Engineering
Ben-Gurion University of the Negev, Beer-Sheva, Israel
moshekr@post.bgu.ac.il,shabtaia@bgu.ac.il

ABSTRACT

This paper presents a study on detecting cyber attacks on industrial control systems (ICS) using convolutional neural networks. The study was performed on a Secure Water Treatment testbed (SWaT) dataset, which represents a scaled-down version of a real-world industrial water treatment plant. We suggest a method for anomaly detection based on measuring the statistical deviation of the predicted value from the observed value. We applied the proposed method by using a variety of deep neural network architectures including different variants of convolutional and recurrent networks. The test dataset included 36 different cyber attacks. The proposed method successfully detected 31 attacks with three false positives thus improving on previous research based on this dataset. The results of the study show that 1D convolutional networks can be successfully used for anomaly detection in industrial control systems and outperform recurrent networks in this setting. The findings also suggest that 1D convolutional networks are effective at time series prediction tasks which are traditionally considered to be best solved using recurrent neural networks. This observation is a promising one, as 1D convolutional neural networks are simpler, smaller, and faster than the recurrent neural networks.

KEYWORDS

Anomaly detection; Industrial control systems; convolutional neural networks

ACM Reference Format:
Moshe Kravchik, Asaf Shabtai. 2018. Detecting Cyber Attacks in Industrial Control Systems Using Convolutional Neural Networks. In *Workshop on Cyber-Physical Systems Security & Privacy (CPS-SPC '18), October 19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3264888.3264896

1 INTRODUCTION

Industrial control systems (ICSs) are widely used and vital to various sectors, including the pharmaceutical industry, manufacturing, and critical infrastructures, such as electricity, water treatment plants, and oil refineries. Historically, these systems ran on proprietary hardware and software in physically secure locations, but more recently they have adopted common information technology (IT) stack and remote connectivity. These changes increase the likelihood of cyber security vulnerabilities and incidents [40]. A number of high impact cyber attacks have been reported in recent years, including the attack on a power grid in the Ukraine in December 2015 [45], the infamous Stuxnet malware that targeted nuclear centrifuges in Iran, and recent attacks on a Saudi oil company [3].

Thus, the ability to detect cyber attacks on ICSs has become a critical task. In this work we focus on an anomaly detection approach, in which we attempt to detect anomalous behavior of the system at the physical level. This approach is based on the assumption that the ultimate goal of the attacker is to influence the physical behavior of the system and aims at protecting the system beyond the network level line of defense. Physical level based anomaly detection can also facilitate early detection and remediation of faulty equipment, which can be of great economical value.

Anomaly detection methods can be based on rules or models of the system [35] [42] [22]. Unfortunately, creating a precise model of complex physical processes is a very challenging task. It requires an in depth understanding of the system and its implementation, which is time consuming and cannot scale well to large and complex systems. An alternative approach which has recently been the focus of interest utilizes machine learning to model ICSs and detect anomalous behavior. A number of studies using supervised machine learning for anomaly detection in ICSs have been published recently [1] [15]. Supervised learning requires labeled training data for normal and attack scenarios, however labeled data for cyber attacks may be difficult to acquire, and this data will naturally not include unknown attack classes. Recently, unsupervised machine learning was shown to be effective [7] [19] for detecting cyber attacks using data obtained from a dedicated water plant testbed (SWaT) [6] which was built to support research related to the design of secure cyber physical systems (CPSs). In [7] the authors use a recurrent neural network (RNN) to detect attacks on a single stage of the six-stage water purification process and report the detection of nine out of ten attacks with four false positives. In addition, [19] compare the performance of a long short term memory (LSTM) based deep neural network (DNN) and one-class SVM to detect attacks on all stages of the same process. They show some improvement, with $F1 = 0.8$ for DNN when measured per log record.

In this work, we present further research on various architectures of unsupervised DNNs to detect cyber attacks on all stages of the SWaT dataset. The main goals of this study are to: (1) identify an efficient deep neural network architecture that can facilitate intrusion detection in ICSs; and (2) improve the cyber attack detection on the SWaT dataset. The contributions of this paper are:

- the successful use of 1D convolutional neural networks (CNN) for detecting anomalies and cyber attacks in ICS data with few false positives;
- a comparison of the efficiency of different neural network architectures for anomaly detection in ICSs which shows that simple 1D CNNs can outperform recurrent networks in this classic sequence prediction task.

2 RELATED WORK

Anomaly and intrusion detection in ICSs (which are a subclass of cyber physical systems) have been extensively studied. A number of comprehensive surveys are dedicated to the classification of techniques and methodologies in this area (e.g., [11] and [33]). According to Mitchell *et al.* [33], ICS anomaly detection methods include knowledge and behavior-based methods. Knowledge-based detection techniques search for known attack characteristics, similar to malware signature techniques in IT intrusion detection. While having low false positive rates, these approaches require maintaining an updated dictionary of attack signatures and are ineffective against zero-day attacks. In contrast, behavior-based techniques search for anomalies in runtime behavior. These techniques are more common in ICS intrusion detection, since ICSs are automated and present more regularity and predictability than typical IT systems. The method used in this study utilizes behavior-based techniques.

Another approach to classifying intrusion detection methodologies is based on the data being monitored. Numerous studies have presented network traffic-based intrusion detection [29]. Ghaeini *et al.* [5] employ this approach on the SWaT dataset used in our study. In our work, we propose an alternative approach based on the data collected from the sensors and actuators, thus focusing on the system behavior at the physical layer.

According to Mitchell *et al.* [33], anomaly detection techniques used in ICS intrusion detection can be broadly divided into supervised, unsupervised, and semi-supervised techniques. Supervised techniques require prior labeling of the system behavior, including the samples of malicious behavior. A precise and representative labeled data is very hard to obtain in practice, and this data is highly dependent on the specific system. Therefore, most recent research on ICS intrusion detection uses unsupervised (learning from unlabeled real data) [16] and semi-supervised (training using a set of partially labeled data, e.g., clean data with no anomalies) approaches. Unsupervised intrusion detection was investigated in [31] which describes a technique based on one-class SVM and k-means clustering. Semi-supervised learning approaches are trained using a collection of "good" data, which is assumed to represent normal system behavior and contain no attacks. While both assumptions should be examined closely, they can be satisfied in many practical situations. Semi-supervised methods usually have lower false-positive rates than their fully unsupervised alternatives [49], [48].

Various machine learning techniques are used in ICS anomaly detection, including support vector machines [31] [15], random forest [1], and artificial neural networks [29], [4]. In the past few years research has been performed applying deep and recurrent neural networks in the task of ICS anomaly detection [32], [7], [19]. In our research 1D convolutional neural networks (1D CNNs) are used and demonstrate stronger attack detection abilities and higher $F1$ scores than previously published papers [7], [19], [28]. Kiranyaz *et al.* [18] use 1D CNNs for anomaly detection in a study aimed at detecting faulty motor bearings based on univariant motor current data. They use 1D CNNs for motor classification as either faulty or good, while in our study we apply 1D CNNs to multivariate time series data and use it to detect multiple instances of cyber attacks. To the best of our knowledge, our work is the first to use 1D CNNs for cyber attack detection in ICSs.

As the aim of our research was to improve upon the attack detection results obtained in previously published studies, we provide a detailed description of this related prior work below. Goh *et al.* in [7] build a three layer LSTM-RNN model that was trained on the data from the first stage of the SWaT dataset. The neural network predicted the expected values of the sensor and actuator data based on their previous values. In order to detect the attacks, the authors applied the cumulative sum (CUSUM) method [34] to the differences between the predicted and actual data. The study results include discovering nine out of ten attacks with four false positives. The authors did not provide detailed data regarding the stability of the results; furthermore, the study was limited to a single stage, and its detection abilities were not cross-validated against other stages and their corresponding attacks. In this study it took 24 hours to train the model; the authors noted that the computational power of the Xeon class server used for the experiments was not capable of carrying out more extensive experiments. In our work we aim to overcome these limitations. First, we identified a neural network architecture that can easily be trained and used it to detect attacks on all of the stages of the SWaT testbed, both individual stages and altogether. This provides us with confidence in the proposed method's ability to detect different kinds of attacks. In addition, our solution scales better and takes less time to train, while having a smaller footprint and less computational demands. Finally, we use a statistical-based anomaly detection method, instead of CUSUM, to overcome the need to fine-tune CUSUM thresholds for every feature, as we explain in Section 5.

Inoue *et at.* [19] performed a comparative study of two methods of anomaly detection: deep neural networks and one-class support vector machines. The study was performed on all six stages of the SWaT dataset and reports a precision of 0.983 and 0.925 for DNN and SVM respectively, with a recall of 0.678 and 0.699 correspondingly. However, a per-attack detection recall for 23 out of 36 attacks was zero for DNN and just slightly better for SVM. The DNN method of anomaly detection, which also utilized LSTM neural network takes a different approach than the one used by Goh *et al.* The authors use a complex architecture that treats sensors and actuators differently, in which the outputs of the LSTM layer are used to predict the outcome of the actuators. The predictions are mixed with actual values and are fed into a fully connected hidden layer to predict a mean value and variance of the first sensor. This process is repeated for the rest of the sensors, and then the sum of the log probabilities of the actuator positions and sensor values provides the outlier factor used for anomaly detection. The resulting network architecture is quite complex, difficult to understand, and resource demanding. In addition, it seems that the neural network requires adaptation to the specific ICS architecture, unlike the solution of Goh *et al.*, which did not require different treatment of sensors and actuators or adjust the number of layers to the number of features in the dataset. Training the model required GPU cluster machines equipped with 10 cores of Intel Xeon E5-2630Lv4, 256GB of RAM, and 8 NVIDIA Tesla P100s, and took up to two weeks for the DNN model. It must be noted that the SVM model did not require such elaborate architecture and only required training on a single machine for a period of 30 minutes. That said, the SVM model was less precise and had 16 attacks with a recall of zero. In our work we aimed to improve upon these results, in terms of simplicity, resource demand, training speed, and attack detection. Our main tool for achieving this goal was a simpler and more efficient 1D CNN architecture.

Lin *et al.* [28] uses a novel graphical model-based approach for anomaly detection. This research shares some common characteristics with ours, namely learning different stages separately and combining the results, and a window-based approach to attacks detection. The authors detect the attacks by using a process that starts with separating the stages into submodels of the related sensors and actuators. The signals in each submodel are subject to different preprocessing dependently on their signal dynamics and are further transformed into symbolic representation. The transformed signals of sensors and actuators are passed to a timed automata and Bayesian network. In addition, the values of sensors and actuators are checked to see whether they exceed the thresholds. An attack is detected by combining of all three detection mechanisms. Compared to this work, our study uses a single detection mechanism (1D CNN) and treats all sensors and actuators uniformly. A comparison of the preliminary detection results shows that our method provides slightly better attack detection scores, though a more thorough comparison of the scoring metrics may be required. One significant achievement of Lin *et al.* is the authors' success in producing an explanation for the attack detection outcome, a topic which was not included in our current study but will be addressed in future research.

3 SECURE WATER TREATMENT (SWAT) TESTBED DATASET

The Secure Water Treatment (SWaT) testbed was built by the Singapore University of Technology and Design in order to provide researchers with data collected from a realistic complex ICS environment. Although a detailed description of the testbed and the dataset can be found in [6], we will provide a brief description below.

The testbed is a fully operational scaled down water treatment plant that produces purified water. As shown in Figure 1, the water goes through a six-stage process, each stage (from P1 to P6) is equipped with a number of sensors and actuators. Sensors include flow meters, water level meters, and conductivity and pH analyzers. Actuators include pumps that transfer water from stage to stage, pumps that dose chemicals, and valves that control inflow. The process is not circular, and the water from P6 is disposed of. The sensors and the actuators at each stage are connected to the corresponding PLC (programmable logic controller), and the

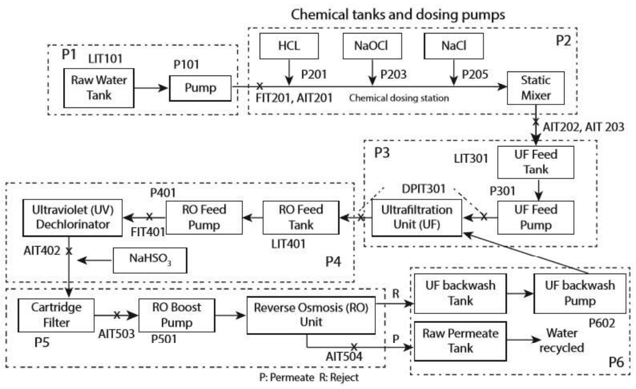

Figure 1: SWaT testbed process overview [6]

PLCs are connected to the SCADA (Supervisory Control And Data Acquisition) system as shown in Figure 2.

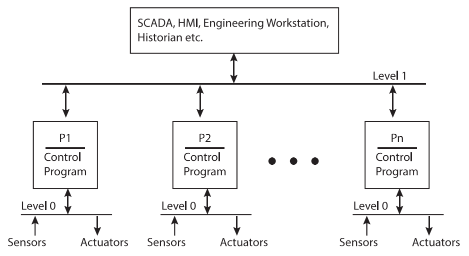

Figure 2: SWaT testbed network overview [6]

The data from all of the sensors and actuators is logged every second in an Historian server, and this data is used for training and testing the system. The dataset contains seven days of recording under normal conditions and an additional four days of recording during which 36 attacks were conducted. The threat model used in the experiment simulated a system that was already infected by attackers who then interfere with normal system operation and spoof the system state to the PLCs causing erroneous commands to be issued to the actuators. The attacks were conducted by altering the network traffic in the level 1 network, spoofing the sensors' values and issuing fake SCADA commands. Attacks include attacks that target a single stage of the process and attacks acting simultaneously at different stages. A table listing the attacks and the corresponding attack times, attack points, and expected and factual outcomes is provided in [6]. For example, attack 21 aims at overflowing the tank at the P1 stage. For that purpose, the value of the water level sensor LIT101 is fixed at 700mm, while the motorized valve controlling water inflow is kept open for 12 minutes. Figure 3 shows the attack, its influence on the LIT101 sensor, and the time it takes the system to stabilize.

It should be noted that the attacks were usually not stealthy, i.e., when a command was issued to the actuator and the actuator changed the system state, the change was not hidden by the attackers. The entire dataset contains 946,722 records, labeled as either

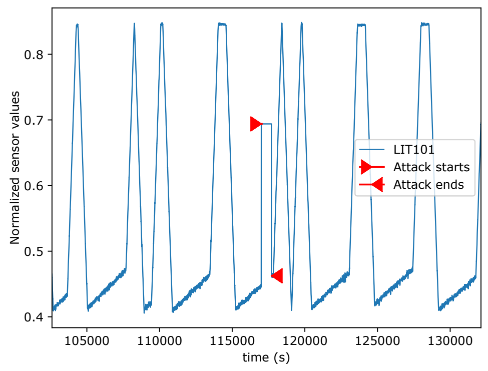

Figure 3: Attack 21 on the LIT101 sensor

attack or normal, with 51 attributes corresponding to the data from sensors and actuators.

4 BACKGROUND ON NEURAL NETWORKS

4.1 Recurrent Neural Networks

This section provides a brief introduction to recurrent neural networks (RNNs), for a detailed discussion, please refer to Lipton *et al.* [30]. Traditional feedforward neural networks assume independence among samples, resetting the state after each sample. This approach has shown strong results for many tasks but has limited success when the samples are related in time, such as audio signals, speech, video, and many other important areas. The main difference between RNNs and standard feedforward neural networks lies in the RNNs' ability to maintain the state between the inputs. While very powerful, RNNs were found to be hard to train due to a vanishing and exploding gradient problem caused by back-propagations across many time steps. This problem has been addressed by a special type of RNNs, long short-term memory (LSTM) networks [17] as well as the use of the truncated back-propagation through time (TBTT) technique [44]. We employed both techniques in our work.

4.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are feedforward neural networks that became popular in image processing following the groundbreaking work of LeCunn *et al.* [27]. CNNs significantly increase the efficiency of neural networks by applying *convolutions*, which are basically filters, to small regions of the input instead of performing matrix multiplication over the entire image at once.

While CNNs used in image processing are two-dimensional (2D), 1D CNNs exist, and they can be successfully used for time series processing, because time series have a strong 1D (time) locality which can be extracted by convolutions [26]. In this work we show that 1D CNNs can be effectively used for detecting cyber attacks in complex multivariate ICS data.

5 ANOMALY DETECTION METHOD

The following statistical window-based anomaly detection method was used in our research. First, a neural network model is used to predict the future values of the data features based on previous values. We provide the model with a sequence $(x_0, x_1, \dots, x_{n-1})$ to predict a sequence $(y_n, y_{n+1}, \dots, y_{n+m})$ where x_i and y_i are input and output vectors of feature values at time i.

The model cannot precisely predict the behavior of the system, which is non-linear and dependent on many factors, some of which are environmental and others of which are dictated by the controller program logic. Moreover, it was observed in our study that specific features can be predicted better than others. Therefore, in order to detect anomalies we chose the following approach.

We calculate the absolute difference between the expected $\hat{y_t}$ and observed y_t values for each feature at time t as follows:

$$\vec{e_t} = \left|\vec{y} - \vec{\hat{y}}\right|. \tag{1}$$

μ_e and σ_e (the mean and standard deviation of the prediction error) are calculated over all of the data. At test time, for each prediction, we calculate the absolute value of the difference between the prediction and the observation and normalize it using the mean and standard deviation for each feature. Then, we effectively have a z-score of the probability of the prediction error for each feature:

$$\vec{z_{e_t}} = \frac{\left|\vec{e_t} - \mu_e\right|}{\sigma_e}. \tag{2}$$

A threshold for the z-scores is used for the anomaly/attack detection. If the maximal value across all of the features exceeds this threshold, we enter the anomaly detection state as follows:

$$\max \vec{z_{e_t}} > T. \tag{3}$$

However, due to irregularities and abrupt state changes in some features (e.g. pump on/off state) such deviations can exist but do not signify a cyber attack event. In order to reduce the number of false alarms, the threshold T must be maintained for at least a specified duration of time, i.e., a time window denoted by W. The threshold T and minimal time window W are hyperparameters for the algorithm and are empirically determined, as will be described later. T basically defines the confidence level in the result. The final decision for an anomaly A_i at time i is determined by the following equation:

$$A_i = \prod_{t=i-W}^{i} \max \vec{z_{e_t}} > T. \tag{4}$$

If the detected anomaly intersects with the attack period extended by a fixed period of additional time immediately following the attack, we consider the attack to be detected. The reason for adding the extra time is the fact that the attacks usually leave an impact on the system and the system's anomalous behavior occasionally continues long after the attack has been performed.

We evaluate the detection performance based on the number of attacks successfully detected (attack-based scoring). We have found this detection method superior to the one used in [19], where the authors referred to the number of correctly detected log records. The best way to explain our choice is by considering an example. Figure 4 shows the water level reported by the LIT101 sensor during and after an attack targeting the first stage of the system and aiming to underflow the first water tank (attack number 30 in the SWaT dataset).

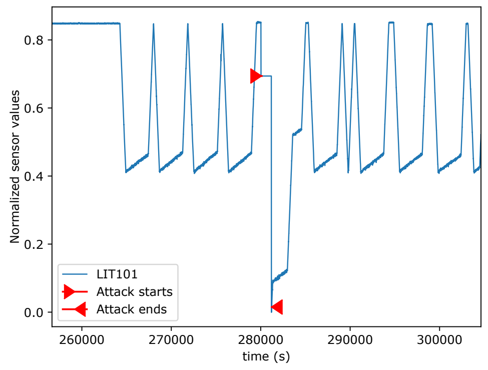

Figure 4: Attack 30 and its impact on the system

In order to achieve the attack goal, the water level reported by the sensor is fixed at 700mm, and the pump that empties the tank is continuously on. Figure 4 clearly shows a recovery period following the actual attack has taken place. During this recovery period the system is not behaving normally, and thus detecting this time period as anomalous is correct. If the detection performance is measured by correctly detecting log records, then the records from the recovery period classified as anomalous would be considered false positives, as there was no attack at this time. We believe that attack-based scoring reflects the dynamics of ICSs under attack better than its log record-based alternative. Another reason for our choice is the observation that the some of the attacks are much longer than others and detecting a single long attack while failing to detect all of the other attacks will still produce high record-based recall. As in real systems, the importance of detecting all attacks is paramount, therefore, we consider attack-based scoring preferable for our study. However, in order to compare our results with prior work we have also presented record-based scores.

The CUSUM algorithm [34], an alternative approach to anomaly detection, was used in [7]. CUSUM is a classic algorithm for change detection in time series data. In CUSUM, the high and low cumulative sums, *SH* and *SL* respectively, are calculated using the following regressive equation:

$$SH_0 = SL_0 = 0$$

$$SH_{n+1} = \max(0, SH_n + x_n - \omega_n) \tag{5}$$

$$SL_{n+1} = \min(0, SL_n + x_n - \omega_n).$$

where x_n denotes the n-th sample of the monitored process, and ω_n is a weight assigned to the sample. In order to determine whether an anomaly has occurred, the authors of [7] compare the *SH* and *SL* to the *upper control limit (UCL)* and *lower control limit (LCL)* correspondingly. If $SH > UCL$ or $SL < LCL$ an anomaly is detected. The UCL and LCL were defined empirically for each feature. This algorithm was tested in our research as well, but we did not use it, because the statistical method produced better results, and the proposed CUSUM method requires choosing more hyperparameters (two per feature instead of just two).

6 EXPERIMENT AND ANALYSIS

6.1 Setup

Model training and testing were performed on an Intel i7-6700K workstation with 32GB of RAM using a NVIDIA 8GB 1080 GPU. With 20% of the training data withheld for validation, training was performed until the loss function calculated for the validation data stopped decreasing or until a training iteration maximum of 100 epochs was reached. The models were implemented using Google's TensorFlow framework version 1.4 [9]. Specifically, we used the low level API which allowed for fine-grained control of the network architecture.

6.2 Data Preprocessing

Since this research focuses on physical layer attacks, we only used the "Physical" subdirectory containing the data collected from the sensors and actuators each second. The network traffic recording that is also present in the dataset was not utilized. The data collected under normal conditions has 496,800 data records and the data collected while performing attacks has 449,919 records with 36 attacks among them. The attacks usually span hundreds of seconds, with the shortest one being just 100 seconds, the longest one taking 35,899 seconds, and the second longest attack taking 1,689 seconds. The first 16,000 records of the training data were trimmed, as the system was unstable (see Figure 5).

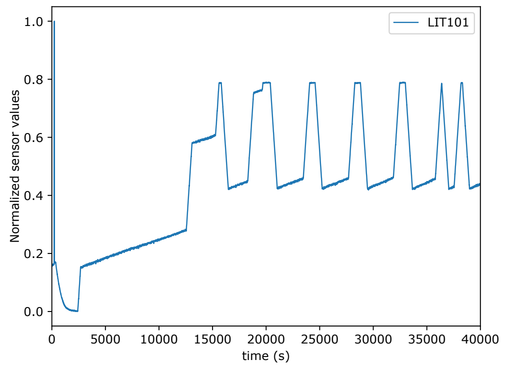

Figure 5: Growth of the water level measured by the LIT101 sensor during the initial warm up period

In addition, the data was normalized to (0,1) scale. The minimal and maximal values of features from the training set were saved and used for scaling the test set. It should be noted that in our final setup we scale both *contiguous* (e.g., water level) and *categorical* (e.g., motor valve closed) values to the (0,1) scale. This generalizes our architecture to different data types.

Several data augmentation techniques were tested, but only one produced an increase in detection score. Specifically, in order to enrich the data with higher-order features, the data is concatenated with newly engineered features representing the *difference* between the current value of the feature and a past value with the given time lag. This difference carries the approximation of the derivative of the feature at the given moment of time as follows:

$$\vec{x}_t = \vec{x}_t | (\vec{x}_t - \vec{x}_{t-lag}). \tag{7}$$

The final transformation we apply to the data is to divide it into batches in order to apply mini-batch learning as the TensorFlow API suggests. Several prediction mode alternatives were tested. One was a sequence-to-sequence model where n timesteps of input were used to predict m timesteps of output, where

$$1 <= m <= n, \tag{8}$$

so that the sequence $(x_i, x_{i+1}, \ldots, x_{i+n-1})$ is used to predict a sequence $(y_j, y_{j+1}, \ldots, y_{j+m})$ where x_i and y_j are input and output vectors of feature values at times i and j correspondingly, and

$$i < j < n + 1. \tag{9}$$

In other words, the predicted sequence is at most as long as the input sequence; the predicted sequence starts after the input one and either overlaps with it or follows it strictly. Better results were achieved by predicting a single vector (i.e., sequence-to-vector prediction), so that

$$\begin{gathered} m = 1 \\ j = n. \end{gathered} \tag{10}$$

The prediction mode described above necessitates that the data batches be extended in order to be able to predict all of the output values. This is best explained with an example. Suppose there are ten points split into two batches, each with five points. Thus, the first batch of five points is used to predict the first point in the second batch. Now consider the last four points of the second batch. They are not predicted, as there are not enough points at the end of the first batch to predict them. With 100 batches of 200 points each, there will be 19,900 points that the model hasn't been trained on. In order to solve this problem, the first batch is extended, appending the first four points of the second batch to the end of the first batch so that they can be used to predict points six to nine in our example, which is presented in Figure 6.

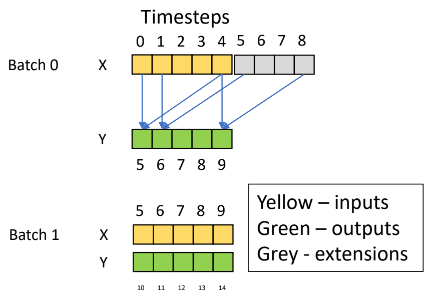

Figure 6: Batches and their extension

6.3 Architectures Used

We began our experiments with recurrent neural networks, a popular neural network architecture approach for processing time series data [30]. We fed the network a sequence of length $n + 1$, passed it through a number of stacked LSTM layers, and applied a fully connected layer at the end to produce the prediction. Initially, we predicted the time window following the input, e.g., feeding the first 200 seconds of data in order to predict the following 200 seconds. In later experiments we followed a different pattern, using only the last value of the output sequence as a prediction for the single data point immediately following the input sequence, i.e., second 201 in our example. We used AdamOptimizer [24] with learning decay for all experiments and the mean squared error (MSE) loss function as follows:

$$MSE = \frac{1}{n} \sum_{t=1}^{n} Y_i - \hat{Y}_i \tag{11}$$

We experimented with multiple learning rates and learning decay strategies. The learning rates were usually in the range of 0.001 to 0.00001, and the decay rate ranged from 0.9 to 0.99. We tested various depths of LSTM layers (from one to ten), state sizes ranging from 64 to 2048, and sequence lengths between 50 and 1000. In addition to the basic LSTM recurrent network, we experimented with gated recurrent cells (GRU) [2] and with bidirectional recurrent neural network architecture [36] based on LSTM cells. Another recurrent network architecture tested was an encoder-decoder LSTM network, based on the ideas in [39] and [2]. We encoded the input sequence using layers of LSTMs; then we used the *state* of the last encoder layer as an input to a stack of LSTM decoders' layers that predict the output sequence.

The main focus of our study was applying convolutional neural networks, and more specifically 1D CNNs, which can be seen in Figure 7 in various architectures.

The most basic architecture follows the classical convolution-ReLU-MaxPooling scheme, where convolutions are 1D and applied to each feature separately along the time axis. We experimented with kernel sizes ranging from two to four and different filter sizes. Typically, we doubled the filter size with each layer. The stack of convolutional layers is followed by a fully connected layer that predicts the output. We used a dropout [38] layer to prevent overfitting. In an attempt to improve the performance of the network, we also experimented with batch normalization as described in [20]. The batch normalization layer goes between the convolutional and ReLU layers. In addition to the classic CNN architecture (CONV-RELU-POOL), we tested other popular CNN architectures. Specifically, we replaced the basic CONV-RELU-POOL block with $(CONV\text{-}RELU) \times N\text{-}MAXPOOL$ as found in VGG network[37]. We also experimented with replacing the convolutional layers with inception layers following the original architecture described in [41]. Inception layers are known to provide superior performance while keeping computational cost low. We used the implementation in [14] as a reference for the inception layer. One of the most recent advances in CNN architecture is the use of residual networks or ResNet [12], and we used a reference implementation of the more recent version of ResNet [13]; in our experiments ResNet blocks provided poorer convergence than other convolutional architectures. Finally, we tested a combined architecture, in which we processed the data by a stack of convolutional (or inception) layers and passed the output to LSTM layers which made the prediction.

6.4 Hyperparameter Tuning

First, we performed our tests on the data from the first stage (P1) of the six-stage process. As P1 has only five features we were able to obtain the results relatively quickly. In addition, it should be easier to detect anomalies in the behavior of the first stage, because at this point in the process as there are no prior stages influencing the internal system state. Our tests had a large parameter space that

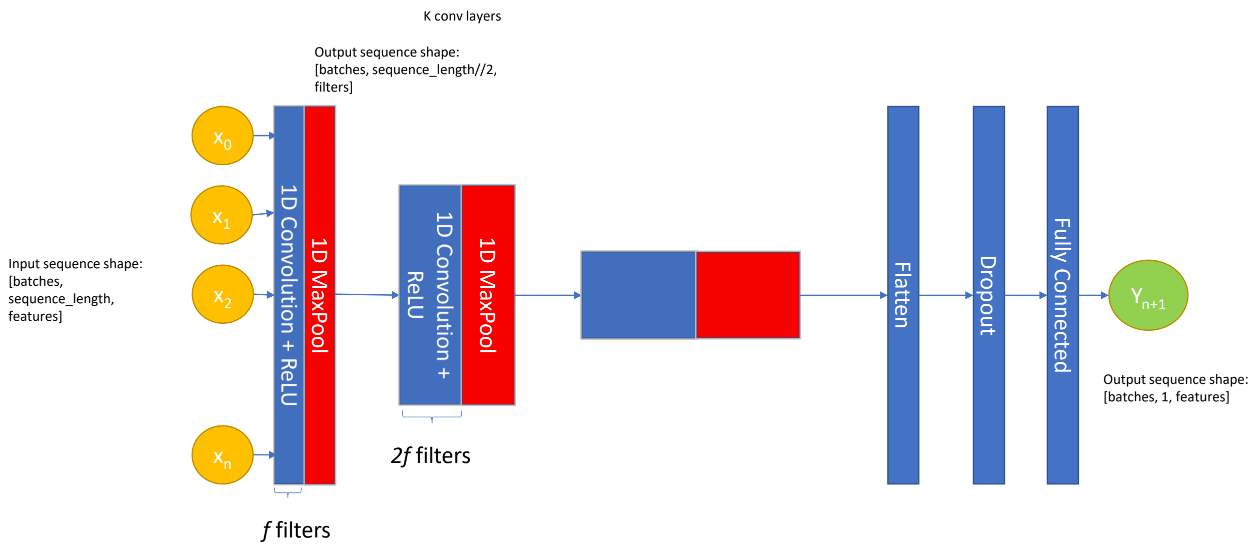
K conv layers
Output sequence shape: [batches, sequence_length//2, filters]
Input sequence shape: [batches, sequence_length, features]
x0
x1
x2
xn
1D Convolution + ReLU
1D MaxPool
1D Convolution + ReLU
1D MaxPool
Flatten
Dropout
Fully Connected
Yn+1
Output sequence shape: [batches, 1, features]
2f filters
f filters

Figure 7: 1D convolutional neural network model

included: input sequence length, network internal state size, output sequence length, the lag between input and output, network depth, dropout value and type (variational vs. regular), CNN kernel size and number of filters, and pooling algorithm and size. We used a grid search strategy to explore the hyperparameters. We selected the $F1$ score as our performance metric; $F1$ is calculated as follows:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{12}$$

Precision and recall are calculated based on the *attack detection* rate according to the anomaly detection algorithm described in Section 5. Precision is the ratio of correctly detected attacks and all detected attacks, and recall is the ratio of correctly detected attacks and all attacks in the data set. We chose to use $F1$ score of all possible F_β scores, as we place equal importance on avoiding false positives and false negatives. While conducting the experiments, we discovered that the detection results vary for multiple runs of the same configuration. This is due to the low number of attacks and the resulting significant impact of a single incorrectly predicted attack. Therefore, we present average scores for multiple (usually five to 10) runs of the same configuration. As explained in Section 5, we use the neural network to predict the output and compare it to the actual output, detecting attacks using the window and threshold hyperparameters. To determine the optimal hyperparameters for the model and stage, we perform a grid search over time windows from 50 to 300 seconds and thresholds between 1.8 and 3.0, optimizing for the best $F1$ with the highest threshold to ensure higher confidence.

6.5 Results

We first show that the selected neural network can learn the system features and predict them with adequate precision. We found that given enough computational capacity, all of the neural networks were able to achieve a root mean square error (RMSE) within a range of 0.02 or less after a small numbers of training epochs. By "enough computational capacity" we refer to the size of the internal state for the LSTMs, the number of filters in the CNNs, and the number of layers in both. We found that this level of error can usually be achieved with two or more layers. Increasing it further does not greatly improve the model accuracy, although some architectures converged faster than others. This might be due to the fact that we used sequences of 256 steps which get to the dimensions of $1 \times filters$ when processed by eight layers. Therefore, we provide the results of at most 8 layers.

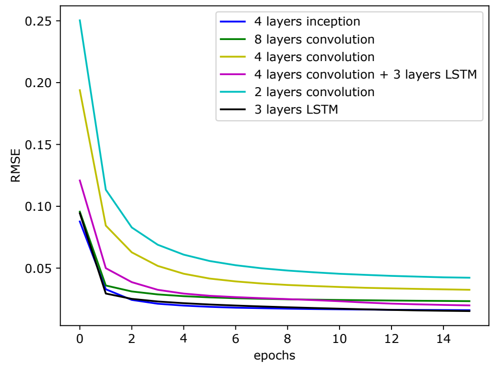

Figure 8: Training errors

As shown in Figure 8, LSTMs and inception-based convolution converge the fastest and produce the lowest training error rate, but other configurations were not far behind and produce similar results with more iterations (only the first 16 iterations are shown).

As seen in Figure 9, validation errors also consistently decreased, converging to the same error range. The test error rate (Figure 10) was also small, however LSTM-based models showed better convergence than pure CNNs, except for the eight layer CNN, which

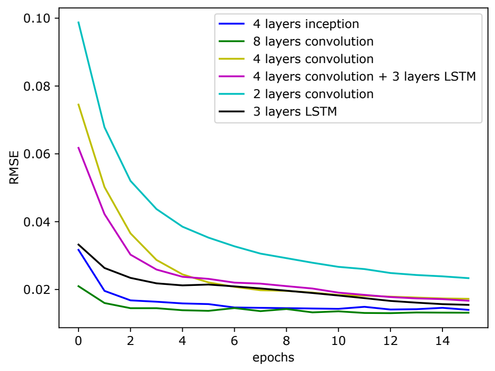

Figure 9: Validation errors

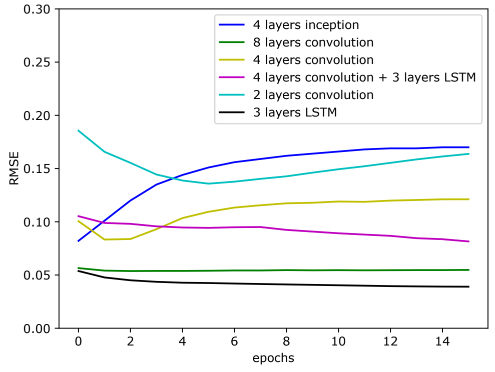

Figure 10: Test errors

had a stable test error level. We also estimated the AUC of the selected anomaly detection method based on per record detection metrics. The reason for using per record labels for the AUC is the ease of defining the number of negative instances, which is problematic in the case of per attack detection (it is impossible to measure how many non-attacks there are). As Figure 11 shows, the anomaly detection algorithm provides a high AUC, reaching 0.967 for the eight layers CNN.

It is interesting to compare the training and test times, as well as different model sizes, as can be seen in Figures 12 - 14. The figures show average times per epoch, as measured at the workstation machines described in Section 6.1. The training and testing times of convolutional networks were shorter by a factor 10 to 20 for testing and 15 to 40 for training, compared to a pure LSTM network. The training and testing times of a convolutional network with additional LSTM layers were also significantly shorter than those of a pure LSTM network, as the convolutional layers significantly decreased the length of sequence input for the recurrent layers. As for the model size, adding more convolutional layers leads to significantly larger models, given our decision to double the number of filters in each layer. These large models were still very fast to train and test. However, we were able to double the number of filters every other layer without performance degradation. We present

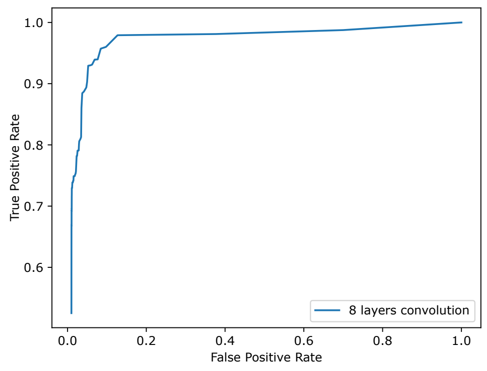

Figure 11: ROC curve for eight layer CNN

the size of an eight layer network built while doubling the number of filters every even layer.

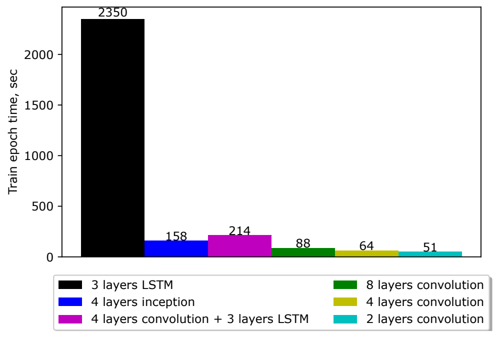

Figure 12: Training times (one epoch)

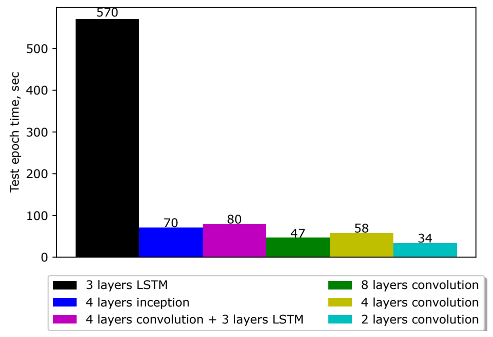

Figure 13: Testing times (one epoch)

Next, we present the attack detection performance comparison at individual stages. The results presented are averaged over 5-10 runs of the same configuration. As there are many hyperparameters, we only present the best combination for each configuration. As Figure 15 shows, pure CNN networks demonstrated better anomaly detection results than their LSTM alternatives. We can also see that inception layers had no advantage over simple convolutions.

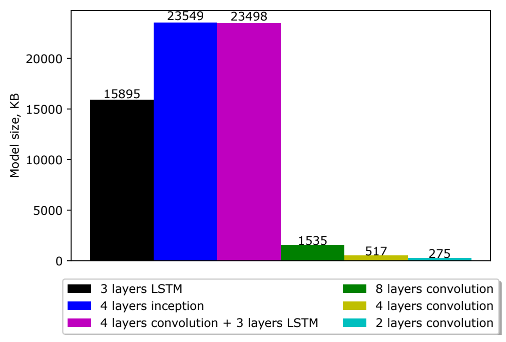

Figure 14: Model size

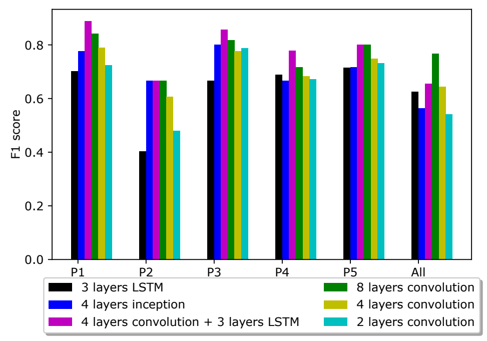

Figure 15: F1 scores per stage

The best results were achieved with the eight layer convolutional network. We omitted the P6 stage from our experiments, as some of the sensors and actuators were not used for data collection; in addition, there is only one attack that involves this stage, thus its detection results are not representative. Other research using this dataset [28] also excluded P6 from their results.

In order to determine which attacks were not discovered by our detection method, we took an ensemble approach and combined the detection results for individual stages. In the ensembled results we discovered that only five attacks were consistently undetected: attacks 4, 13, 14, 29, and 35. When reviewing the description of these tests we found that attacks 4, 13, 14, and 29 failed to produce the expected impact on the system. For example, attack 29 aimed to waste chemicals by opening three dosing pumps simultaneously. However, due to some mechanical interlock, the pumps did not start. As these attacks did not have a physical impact, it is reasonable that they were not detected by our model. We also observed that attack 4 was carried out on the MV504 motorized valve, while the description of the SWaT testbed does not contain such an element. Attack 35 involved turning off both pumps for the P1 stage simultaneously, however it seems that the model did not consider this to be an attack as there are many time periods when both pumps were off in the training data.

When combining the attacks detected in individual stages, the model successfully detected 31 of 36 attacks, thus improving upon the results reported in [19] and [28]. The $F1$ score of the ensemble of the eight layer 1D CNN model was 0.886, with precision of 0.912 and recall of 0.861. We also observed that when trained on the data of all of the stages together, the model achieves a slightly lower $F1$: 0.767 for the eight layer network. We estimate that the reason the $F1$ of the single combined model is lower than the $F1$ of the ensemble of the individual stages is that the proposed architectures were too small to capture all of the complex dependencies across different stages, including time delays between these dependencies. Also, modeling a single stage separately allows the network to better capture the dependency between the features introduced by the PLC logic, for example turning off the pump when the water in the tank reaches its preset lower limit.

In contrast to performing per attack calculations, when calculating the score per log record detection, the $F1$ for the model trained on the data of all of the stages together was 0.871 for eight layers, thus improving upon previously published results by [19] and [28]. Tables 1 and 2 summarize the averaged results of the best working network configuration presented in the figures above. The main hyperparameters of the configurations appear in the tables.

In order to make a comprehensive comparison with previous work, a recall evaluation per attack scenario was conducted, and the results of this evaluation are presented in Tables 3 and 4. The tables are based on the tables that appear in [28] and add to them the results of our study. The DNN and SVM data originates from the research of [19] and the TABOR data comes from the research of [28], both of which are described in Section 2. The comparison shows that a 1D CNN provides recall improvement in most of the cases.

7 CONCLUSIONS AND FUTURE WORK

This section summarizes the main results of our study and discusses its limitations, adversarial threats and countermeasures. The section concludes with possible future research.

In this paper, we studied the use of deep learning architectures, namely convolutional and recurrent neural networks for cyber attack detection in ICSs. We tested a number of architectures and hyperparameters and proposed a statistical window-based anomaly detection method that has been shown effective in detecting the attacks in the SWaT dataset. We introduced a model based on a 1D convolutional neural network which successfully detected 31 of 36 attacks. Its superior run time and performance have been elaborated upon and show great promise for ICS cyber attack detection. Beyond achieving our goal of developing an efficient and precise method of detecting attacks in the SWaT dataset, this work's novelty is in utilizing 1D CNN networks for this task. While convolutional networks are not new, recent research considers the classification and prediction of sequence and series data to be performed best by recurrent architectures, and there are numerous studies [23] [10] and books [8] suggesting that recurrent networks cannot be beaten in these tasks. Therefore, it was natural to address the problem using RNN and LSTM; the finding that 1D CNNs may do the job better may reflect a more general observation, namely that 1D CNNs can outperform recurrent architectures in complex multivariate time series prediction.

In order to investigate this idea further, we tested a 1D CNN model with a popular sequence prediction benchmark: sequential

Table 1: Test Results

Network configuration	Train epoch time, sec	Test epoch time, sec	Model Size, KB
4 inception layers, kernel = 2, 32 filters	158.69	70.8	23549
8 convolutional layers, kernel = 2, 32 filters	88	47	1585
4 convolutional layers, kernel = 2, 32 filters	64	58	517
4 convolutional layers + 3 LSTM layers, kernel = 2, 32 filters	214	80	23498
2 convolutional layers, kernel = 2, 32 filters	51	34	275
3 LSTM layers with state = 256	2350	570	15895

Table 2: $F1$ Scores Per Stage

Network configuration	P1	P2	P3	P4	P5	All
4 inception layers, kernel = 2, 32 filters	0.777	0.666	0.8	0.666	0.716	0.564
8 convolutional layers, kernel = 2, 32 filters	0.842	0.667	0.8	0.717	0.8	0.767
4 convolutional layers, kernel = 2, 32 filters	0.79	0.607	0.776	0.683	0.748	0.644
4 convolutional layers + 3 LSTM layers, kernel = 2, 32 filters	0.888	0.666	0.857	0.778	0.8	0.655
2 convolutional layers, kernel = 2, 32 filters	0.725	0.48	0.787	0.672	0.731	0.542
3 LSTM layers with state = 256	0.701	0.403	0.666	0.689	0.714	0.626

Table 3: Attack detection performance comparison. Results for 8 layers 1D CNN are shown.

Method	Precision	Recall	F1
DNN	0.983	0.678	0.803
SVM	0.925	0.699	0.796
TABOR	0.862	0.788	0.823
1D CNN combined records	0.968	0.791	0.871
1D CNN combined attacks	0.958	0.639	0.767
1D CNN ensembled records	0.867	0.854	0.860
1D CNN ensembled attacks	0.912	0.861	0.886

MNIST (used by Le *et al.* in [25] and since then by many others [47] [46] [21]). A simple six layer 1D CNN network with 20 filters and a kernel size of 10 (125K parameters) achieved an accuracy of 1.0 for sequential MNIST and 0.982 for permuted MNIST. This accuracy is very high, and while it does not surpasses the state of the art, it was achieved without any hyperparameter or network architecture tuning. This finding might be of a significant value, as 1D CNNs have a number of advantages over RNNs: they are much simpler, orders of magnitude faster, smaller, and well understood. CNNs are also advantageous in parallelized computation and have more stable gradients.

In the current research we found that having a dedicated model for each stage and creating an ensemble based on their outputs produces better results than having a single model for the whole system. This may be due to limitations of the hardware used for the experiments which did not allow building a model large enough to represent all of the stages and their dependencies (including time dependencies) together. In addition to higher results, training separate models for stages will scale better than a single model, but the ways to learn interstage dependencies need to be examined. The research performed was not based on specific assumptions regarding the system modeled and its attacks, and thus we expect the results to be generic. At the same time, one potential threat to the validity of the research is the fact that it was tested on one dataset from one type of industrial process. Another potential threat to the validity is the fact that the attacks were relatively simple and there was no attempt to disguise the effects of the attacks, neither of which can be assumed in real-life scenarios.

Adversarial attacks can be carried against the proposed method. One such threat is an adversary that injects attacks during the training period of the algorithm, so that the model learns them as a normal behavior. One potential approach to its solution is cross-referencing models from similar environments. Another threat to the proposed algorithm is a replay attack, where the adversary succeeds in spoofing the values of all of the sensors in the attacked area while alternating the ICS behavior. One possible countermeasure to this threat is based on building a model for a particular subsystem with the addition of data from related sensors from adjacent subsystems. Due to physical influences of the industrial processes, a subsystem's behavior affects other connected subsystems; thus, the malicious behavior will be reflected by deviations in the behavior of other subsystems. To overcome this, the adversary would need to mount a coordinated attack on multiple subsystems, which is a much harder task.

Beyond addressing the above mentioned limitations this research can be expanded in a number of directions:

- studying methods for embedding higher-order process modeling for anomaly detection,

Table 4: Point recall evaluation comparison.

No.	Scenario No.	Description of Attack	DNN	SVN	TABOR	1D CNN Records	1D CNN Attacks
1	1	Open MV-101	0	0	0.049	0.995	1
2	2	Turn on P-102	0	0	0.930	1	1
3	3	Increase LIT-101 by 1mm every second	0	0	0	0.225[1]	0.9
4	4	Open MV-504	0	0.035	0.328	0	0
5	6	Set value of AIT-202 at 6	0.717	0.720	0.995	0.903	1
6	7	Water level LIT-301 increased above HH	0	0.888	0	1	1
7	8	Set value of DPIT as <40kpa	0.927	0.919	0.612	1	1
8	10	Set value of FIT-401 at >0.7	1	0.433	0.994	1	1
9	11	Set value of FIT-401 at 0	0.978	1	0.998	1	1
10	13	Close MV-304	0	0	0	0	0
11	14	Do not let MV-303 open	0	0	0	0	0
12	16	Decrease water level LIT-301 by 1mm each second	0	0	0	0.236	1
13	17	Do not let MV-303 open	0	0	0.597	0.631	1
14	19	Set value of AIT-504 at 16 uS/cm	0.123	0.13	0.004	0[1]	1
15	20	Set value of AIT-504 at 255 uS/cm	0.845	0.848	0.997	1	1
16	21	Keep MV-101 on continuously; set value of LIT-101 at 700mm	0	0.0167	0.083	0.907	1
17	22	Stop UV-401; set value of AIT502 at 150; force P-501 to remain on	0.998	1	0.998	1	1
18	23	Set value of DPIT301 at >0.4 bar; keep MV302 open, P602 closed	0.867	0.875	0	1	1
19	24	Turn off P-203 and P-205	0	0	0	0.169	1
20	25	Set value of LIT-401 at 1000; P402 is kept on	0	0.009	0	0.019	0.2
21	26	P-101 is turned on continuously; set value of LIT-301 at 801mm	0	0	0.999	1	1
22	27	Keep P-302 on; set value of LIT401 at 600mm until 1:26:01	0	0	0.196	0.063	1
23	28	Close P-302	0.936	0.936	1.000	1	1
24	29	Turn on P-201; turn on P-203; turn on P-205	0	0	0	0	0
25	30	Turn P-101 and MV-101 on; set value of LIT-101 at 700mm;	0	0.003	0.999	1	1
26	31	Set LIT-401 at less than L	0	0	0	0.303	0.2
27	32	Set LIT-301 at above HH	0	0.905	0	0.935	1
28	33	Set LIT-101 at above H	0	0	0.890	0.883	1
29	34	Turn P-101 off	0	0	0.990	0.6	1
30	35	Turn P-101 off; keep P-102 off	0	0	0.258	0	0
31	36	Set LIT-101 at less than LL	0	0.119	0.889	0.878	1
32	37	Close P-501; set value of FIT-502 at 1.29 at 11:18:36	1	1	0.998	0.895	1
33	38	Set value of AIT402 at 260; set value of AIT502 at 260	0.923	0.927	0.996	0.864	1
34	39	Set value of FIT-401 at 0.5; set value of AIT-502 at 140 mV	0.940	0	0.369	0.908	1
35	40	Set value of FIT-401 at 0	0.933	0.927	0.997	1	1
36	41	Decrease LIT-301 value by 0.5mm per second	0	0.357	0	0.638	1

[1] The attack or a major part of it detected after the attack ended

- investigating the application of recent audio generative network architectures, e.g. WaveNet [43] to anomaly detection,
- applying the proposed anomaly detection method to streaming data and comparing its performance to online anomaly detection algorithms,
- measuring and improving timeliness of the attack detection,
- improving the interpretability of the results, and
- applying the anomaly detection method to faulty ICS equipment behavior detection.

8 ACKNOWLEDGEMENTS

The authors thank iTrust, Centre for Research in Cyber Security, Singapore University of Technology and Design for creating and providing the SWaT dataset. The authors would also like to thank Lihi Shiloh and Dr. Ron Zohar for their assistance and valuable insights at various stages of this research, as well as Rafael Defense Systems for supporting this work.

REFERENCES

[1] Justin M Beaver, Raymond C Borges-Hink, and Mark A Buckner. 2013. An evaluation of machine learning methods to detect malicious SCADA communications. In *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, Vol. 2. IEEE, 54–59.

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[3] ForeignPolicy. 2017. Cyberattack Targets Safety System at Saudi Aramco. (2017). http://foreignpolicy.com/2017/12/21/cyber-attack-targets-safety-system-at-saudi-aramco/ Last accessed March 2018.

[4] Wei Gao, Thomas Morris, Bradley Reaves, and Drew Richey. 2010. On SCADA control system command and response injection and intrusion detection. In *eCrime Researchers Summit (eCrime), 2010*. IEEE, 1–9.

[5] Hamid Reza Ghaeini and Nils Ole Tippenhauer. 2016. Hamids: Hierarchical monitoring intrusion detection system for industrial control systems. In *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*. ACM, 103–111.

[6] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. 2016. A dataset to support research in the design of secure water treatment systems. In *International Conference on Critical Information Infrastructures Security*. Springer, 88–99.

[7] Jonathan Goh, Sridhar Adepu, Marcus Tan, and Zi Shan Lee. 2017. Anomaly detection in cyber physical systems using recurrent neural networks. In *High Assurance Systems Engineering (HASE), 2017 IEEE 18th International Symposium on*. IEEE, 140–145.

[8] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[9] Google. 2016. TensorFlow, An open-source machine learning framework for everyone. (2016). https://www.tensorflow.org/ Last accessed March 2018.

[10] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2017. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems* 28, 10 (2017), 2222–2232.

[11] Song Han, Miao Xie, Hsiao-Hwa Chen, and Yun Ling. 2014. Intrusion detection in cyber-physical systems: Techniques and challenges. *IEEE systems journal* 8, 4 (2014), 1052–1062.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). http://arxiv.org/abs/1512.03385

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity Mappings in Deep Residual Networks. *CoRR* abs/1603.05027 (2016). http://arxiv.org/abs/1603.05027

[14] Burak Himmetoglu. 2017. Time series classification with Tensorflow. (2017). https://github.com/healthDataScience/deep-learning-HAR/blob/master/HAR-CNN-Inception.ipynb Last accessed March 2018.

[15] Raymond C Borges Hink, Justin M Beaver, Mark A Buckner, Tommy Morris, Uttam Adhikari, and Shengyi Pan. 2014. Machine learning for power system disturbance and cyber-attack discrimination. In *Resilient Control Systems (ISRCS), 2014 7th International Symposium on*. IEEE, 1–8.

[16] Geoffrey E Hinton and Terrence Joseph Sejnowski. 1999. *Unsupervised learning: foundations of neural computation*. MIT press.

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

[18] Turker Ince, Serkan Kiranyaz, Levent Eren, Murat Askar, and Moncef Gabbouj. 2016. Real-time motor fault detection by 1-D convolutional neural networks. *IEEE Transactions on Industrial Electronics* 63, 11 (2016), 7067–7075.

[19] Jun Inoue, Yoriyuki Yamagata, Yuqi Chen, Christopher M Poskitt, and Jun Sun. 2017. Anomaly detection for a water treatment system using unsupervised machine learning. *arXiv preprint arXiv:1709.05342* (2017).

[20] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15)*. JMLR.org, 448–456. http://dl.acm.org/citation.cfm?id=3045118.3045167

[21] Li Jing, Yichen Shen, Tena Dubček, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljačić. 2016. Tunable efficient unitary neural networks (EUNN) and their application to RNNs. *arXiv preprint arXiv:1612.05231* (2016).

[22] Austin Jones, Zhaodan Kong, and Calin Belta. 2014. Anomaly detection in cyber-physical systems: A formal methods approach. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE, 848–853.

[23] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*. 2342–2350.

[24] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[25] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. 2015. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941* (2015).

[26] Yann LeCun, Yoshua Bengio, and others. 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* 3361, 10 (1995), 1995.

[27] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. 1990. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. 396–404.

[28] Qin Lin, Sridhar Adepu, Sicco Verwer, and Aditya Mathur. 2018. TABOR: A Graphical Model-based Approach for Anomaly Detection in Industrial Control Systems. (2018).

[29] Ondrej Linda, Todd Vollmer, and Milos Manic. 2009. Neural network based intrusion detection system for critical infrastructures. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*. IEEE, 1827–1834.

[30] Zachary C Lipton, John Berkowitz, and Charles Elkan. 2015. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019* (2015).

[31] Leandros Maglaras, Helge Janicke, Jianmin Jiang, and Andrew Crampton. 2016. Novel Intrusion Detection Mechanism with Low Overhead for SCADA Systems. *Security Solutions and Applied Cryptography in Smart Grid Communications* (2016), 160.

[32] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. Long short term memory networks for anomaly detection in time series. In *Proceedings*. Presses universitaires de Louvain, 89.

[33] Robert Mitchell and Ing-Ray Chen. 2014. A survey of intrusion detection techniques for cyber-physical systems. *ACM Computing Surveys (CSUR)* 46, 4 (2014), 55.

[34] Ewan S Page. 1954. Continuous inspection schemes. *Biometrika* 41, 1/2 (1954), 100–115.

[35] Fabio Pasqualetti, Florian Dörfler, and Francesco Bullo. 2011. Cyber-physical attacks in power networks: Models, fundamental limitations and monitor design. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*. IEEE, 2195–2201.

[36] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[37] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). http://arxiv.org/abs/1409.1556

[38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1929–1958. http://dl.acm.org/citation.cfm?id=2627435.2670313

[39] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. 2015. Unsupervised Learning of Video Representations using LSTMs. *CoRR* abs/1502.04681 (2015). http://arxiv.org/abs/1502.04681

[40] Keith A. Stouffer, Joseph A. Falco, and Karen A. Scarfone. 2011. *SP 800-82. Guide to Industrial Control Systems (ICS) Security: Supervisory Control and Data Acquisition (SCADA) Systems, Distributed Control Systems (DCS), and Other Control System Configurations Such As Programmable Logic Controllers (PLC)*. Technical Report. Gaithersburg, MD, United States.

[41] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. *CoRR* abs/1409.4842 (2014). http://arxiv.org/abs/1409.4842

[42] André Teixeira, Daniel Pérez, Henrik Sandberg, and Karl Henrik Johansson. 2012. Attack models and scenarios for networked control systems. In *Proceedings of the 1st international conference on High Confidence Networked Systems*. ACM, 55–64.

[43] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

[44] Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1, 2 (1989), 270–280.

[45] Wired. 2016. INSIDE THE CUNNING, UNPRECEDENTED HACK OF UKRAINE'S POWER GRID. (2016). https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/ Last accessed March 2018.

[46] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. 2016. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*. 4880–4888.

[47] Saizheng Zhang, Yuhuai Wu, Tong Che, Zhouhan Lin, Roland Memisevic, Ruslan R Salakhutdinov, and Yoshua Bengio. 2016. Architectural complexity measures of recurrent neural networks. In *Advances in Neural Information Processing Systems*. 1822–1830.

[48] Yichi Zhang, Lingfeng Wang, Weiqing Sun, Robert C Green, and Mansoor Alam. 2011. Artificial immune system based intrusion detection in a distributed hierarchical network architecture of smart grid. In *Power and Energy Society General Meeting, 2011 IEEE*. IEEE, 1–8.

[49] Yichi Zhang, Lingfeng Wang, Weiqing Sun, Robert C Green II, and Mansoor Alam. 2011. Distributed intrusion detection system in a multi-layer network architecture of smart grids. *IEEE Transactions on Smart Grid* 2, 4 (2011), 796–808.