SAND2018-8542C

# Physical Security Assessment Using Temporal Machine Learning

Meghan A. Galiardi*, Stephen J. Verzi, Gabriel C. Birch, Jaclynn J. Stubbs, Bryana L. Woo, and Camron G. Kouhestani
Sandia National Laboratories
1515 Eubank SE, Albuquerque, NM, USA
*mgaliar@sandia.gov

***Abstract*—Nuisance and false alarms are prevalent in modern physical security systems and often overwhelm the alarm station operators. Deep learning has shown progress in detection and classification tasks, however, it has rarely been implemented as a solution to reduce the nuisance and false alarm rates in a physical security systems. Previous work has shown that transfer learning using a convolutional neural network can provide benefit to physical security systems by achieving high accuracy of physical security targets [10]. We leverage this work by coupling the convolutional neural network, which operates on a frame-by-frame basis, with temporal algorithms which evaluate a sequence of such frames (e.g. video analytics). We discuss several alternatives for performing this temporal analysis, in particular Long Short-Term Memory and Liquid State Machine, and demonstrate their respective value on exemplar physical security videos. We also outline an architecture for developing an ensemble learner which leverages the strength of each individual algorithm in its aggregation. The incorporation of these algorithms into physical security systems creates a new paradigm in which we aim to decrease the volume of nuisance and false alarms in order to allow the alarm station operators to focus on the most relevant threats.**

## I. Introduction

Physical security systems (PSS) typically rely upon sensing devices (e.g. microwaves, active infrared beam break sensors) to create an alarm, assessment devices (e.g. cameras) to assess alarms and human operators to determine the validity or severity of an event. This traditional architecture places a significant burden on human operators, as all sources of nuisance alarms or false alarms must be evaluated by a human after watching the associated assessment video. This reliance upon human analysts has been shown to potentially create challenging conditions for vigilance maintenance, which ultimately could reduce the effectiveness of the system [9]. New approaches are needed that place intermediary algorithmic components between raw sensed events and human notification.

New advancements in machine learning could aid in solving persistent problems in this domain by:

- Lowering nuisance alarms and false alarms
- Enabling human operators to focus on the most challenging alarms to analyze rather than being inundated with easily assessed alarms
- creating foundational technology the enables the creation of autonomous security systems that ultimately multiples force
- Facilitating new security paradigms such as distributed security or autonomous security

The vision for integrating algorithmic components into a physical security system is as follows: A sensor in the physical security system is alarming. Instead of sending the alarm directly to the alarm station operator, video data leading up to the alarm is sent to an intermediary algorithmic component. The machine learning algorithm will then analyze the data to classify the source of the alarm. If the alarm is classified as as nuisance or false alarm with high confidence, then the alarm is logged, but not displayed to the operator. If the confidence is not as high, a warning may be sent to the operator with the algorithm's best guess at a classification and the operator can review the classification. This interaction can also be used to determine if the machine learning algorithms need to be retrained and can provide the necessary data for retraining. If the alarm is classified as an intrusion with high confidence, then the operator is alerted to the intrusion and the algorithmic components could also assist in determining responses. This intermediary algorithmic component can help lessen the burden of nuisance and false alarms on the operator while allowing them to focus on the real intrusions.

The goal of this paper is to describe an approach for incorporating temporal machine learning algorithms into a physical security system. The paper is organized as follows: Section II describes previous work, both on the physical security side and algorithmic side, Section III describes the architecture the authors have implemented, Section IV discusses the results of various experiments to determine the benefit of such algorithms, and Section V discusses the improvements planned as this research progresses.

## II. Previous work

Not much work has been done on applying machine learning algorithms to physical security. One previous effort has shown success of using transfer learning to assess physical security threats using a video assessment data [10]. This

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525

work extracted individual frames from the video data set and performed a frame-by-frame classification. Our paper seeks to analyze a different approach using temporal machine learning algorithms. The authors conjecture that temporal machine learning algorithms can identify dependencies across time and accurately classify video from a physical security system.

The architecture developed in this work will use three different machine learning algorithms: convolutional neural networks (CNNs), long short-term memory (LSTMs), and liquid state machines (LSMs). The design of CNNs are based on an animal's visual cortex [5]. Groups of nodes in one layer feed into a single node in the next layer known as the visual field. CNNs were first used in 1998 for document recognition [7], but are more commonly known for their extremely good classification of images. LSTMs are a fairly new type of neural network first discovered in 1997 by Hochreiter and Schmidhuber [4]. In the innovative paper, they showed that LSTMs can solve long time lag dependencies that had never previously been solved by a neural network. Shortly thereafter, Gers and colleagues improved upon the LSTM by adding adding functionality which allows an LSTM to learn to reset itself at the end of a data sequence [2]. LSMs are a type of reservoir computing neural network in which nodes in the reservoir compute a dynamical system in time and the state of the dynamical system is read out to perform classification. These types of neural networks were most recently reinvented simultaneously as Echo State Networks and LSMs by Jaeger and Mass, respectively [6], [8]. By leveraging the advances in all three of these algorithms, we seek to create an architecture for classifying physical security data.

## III. Architecture

We design an architecture which has three main components:

1) **Preprocess the data**
2) **Classify videos using temporal algorithms**
3) **Combine the results of the different algorithms**

The structure of our video classification architecture is shown in Fig. 1.

### A. Preprocessing data

The input data for our architecture is physical security videos. A video can be deconstructed into a sequence of images where the sequential aspect represents the temporal nature of the video. As shown in Fig. 2, a single video can be represented as a 3-dimensional object with size $image\ height \times image\ width \times video\ length$ where the first two dimensions are the spatial dimensions of each image and the third dimension is the temporal dimension of the video. However, for the temporal algorithms in this architecture (see III-B), we want to use a 2-dimensional objects with only one spatial dimension followed by the temporal dimension. This forces us to represent each image of a the video in only 1 spatial dimension and this transformation is done by preprocessing the data.

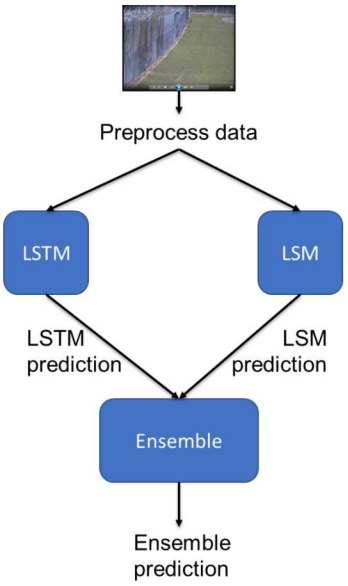


Fig. 1. Temporal architecture for classifying physical security videos

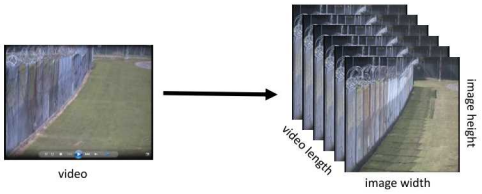


Fig. 2. 3-dimensional representation of video

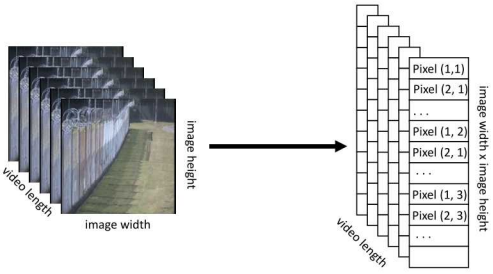


Fig. 3. Preprocessing using vectorization.

The most straightforward way to preprocess a 2-dimensional image into a 1-dimensional representation is to simply vectorize the image, shown in Fig. 3. That is, take all the columns of the image and stitch them together in one large vector. This preprocessing step is fast, but looses most of the spatial relationships in the image.

Spatial relationships in each image are important for good performance of the algorithms. CNNs have been shown to work well in identifying spatial relationships in images by representing these relationships as a set of features. Our second

approach in preprocessing is to use a CNN to extract features from each image, shown in Fig. 4. There exist many CNNs which have been trained on a wide variety of images and show good performance. We therefore choose to use existing pre-trained CNNs as a preprocessing step, namely Inception-v3 [12] and Googlenet [11]. The classification layer of these CNNs is stripped off to obtain a features vector for each image.

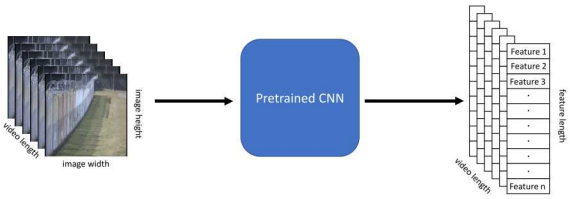


Fig. 4. Preprocessing using CNN feature extraction

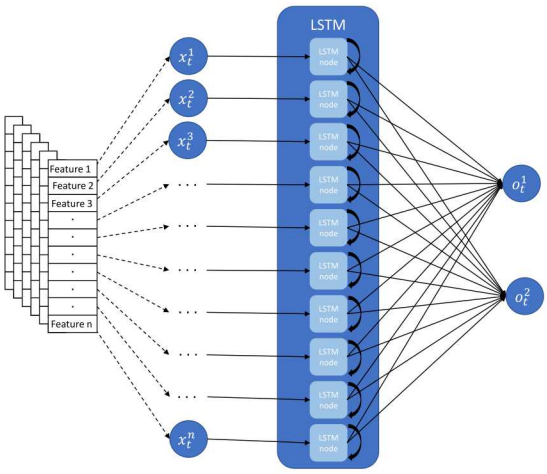


Fig. 5. Architecture used for the LSTM for a 2-class problem

### B. Temporal Algorithms

We chose to start with two temporal algorithms: Long Short-Term Memory (LSTM) and Liquid State Machine (LSM). Both algorithms have shown good results in language processing, handwriting recognition, time-series prediction, among others.

*1) Long Short-Term Memory (LSTM):* LSTMs are a type of recurrent neural network. Recurrent neural networks are similar to feed forward neural networks except the current state of a node is fed back to itself in addition to feeding forward to the next layer of the network. LSTMs can be thought of in the same way as recurrent neural networks, except each node now has a more complex structure. LSTM nodes have additional gates which can add and remove information to the information flow within the sequence. There are three sigmoidal gates which add/remove information, the first of which is a forget gate. A forget gate decides how much of the previous information to keep. If the gate outputs a 1 then it means keep all the previous information while 0 means forget all the previous information and partial information is kept otherwise. The second type of gate is the input gate which determines what values within the information flow will be updated (i.e. what new information we care about). The last gate decides how much information we are going to output to the next node. Each of these gates has their own set of weights and are adjusted via a learning process just like traditional recurrent neural networks. Thus LSTMs learn what information to forget, input, and output. LSTMs maintain a more constant error that is back propagated though time and therefore continue to learn over much longer timescales than traditional recurrent neural networks.

For the architecture developed in this paper, each feature in the features vector of the input has a corresponding LSTM node and the input sequence for that feature is the input to the LSTM node. The output from the LSTMs are then densely connected to a classification layer. Since the input is a sequence, the output from the classification layer is also a sequence, but the final classification is read after the entire input sequence has propagated through the LSTM. The architecture for the LSTM is shown in Fig. 5.

*2) Liquid State Machine:* The input into liquid state machines is given as a vector $x(t)$ which represents a sequence of disturbances to the liquid and the output function $o(t)$ is a function which provides real time analysis of the input such as classification. The goal is to have the LSM learn the function that maps the input function to the output function. The input nodes are randomly connected to the liquid nodes, and the liquid nodes are also randomly connected to each other. At every timestep, the liquid has an internal state denoted $x^M(t)$ which represents the response of the liquid to all the previous inputs $s \leq t$. In other words, the current state of the liquid can be thought of as a filter, $L^M$ applied to the input function

$$x^M(t) = (L^M x)(t).$$

Additionally, LSMs contain a readout function, $f^M$, which transforms the current state of the liquid to the output function

$$y(t) = f^M(x^M(t)).$$

This readout function can be any function, but most commonly used is a linear readout function such as an SVM since SVMs are faster and easier to train than RNNs. Readout functions are most often memoryless in that they only depend on $x^M(t)$ and not $x^M(s)$ for $s \leq t$. The readout function is learned to map the state of the liquid to the output function. Thus there is no training of weights within the liquid, just learning of the readout function.

For the architecture described in this paper, each feature in the features vector is randomly connected to the liquid using a uniform probability distribution and the nodes in the liquid are randomly connected to each other with probability 0.1. The readout function used is a simple one layer dense classification layer. Similar to the LSTM, the output from the classification layer is also a sequence, but the final classification is read after the entire input sequence has propagated through the LSM. The architecture for the LSM is shown in Fig. 6.

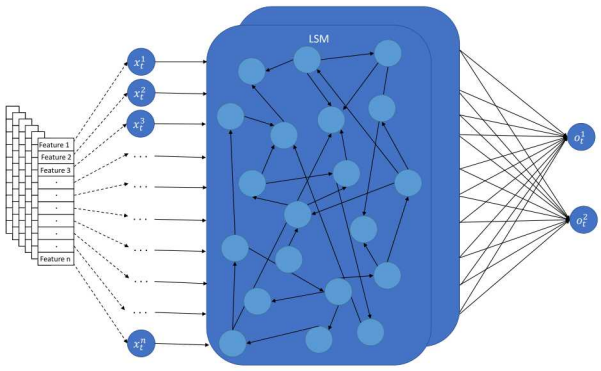


Fig. 6. Architecture used for the LSM for a 2-class problem

*3) Ensemble:* It is anticipated that the LSTM and LSM included in the architecture will have different strengths and weaknesses for physical security data sets. By combining the classification results from both the LSTM and LSM into an ensemble allows for these strengths to be utilized. The ensemble implemented in the architecture is a voting ensemble. Given a video, both the LSTM and LSM will output a vector of probabilities which represents the likelihood that the video is of one of the $m$ classes

$$p_{lstm} = [p^1_{lstm}, p^2_{lstm}, \ldots, p^m_{lstm}]$$

$$p_{lsm} = [p^1_{lsm}, p^1_{lsm}, \ldots, p^m_{lsm}].$$

A voting ensemble weights the class probabilities of the LSTM and LSM and the final classification is the class with the largest classification probability

$$p = w_{lstm}p_{lstm} + w_{lsm}p_{lsm}.$$

For the initial experiments used in this paper, each algorithm had equal voting ($w_{lstm} = w_{lsm} = .5$), however, in the future a learning ensemble will be implemented to better leverage the benefits of each model (see section V).

## IV. Experimental Results

The data used for the following results is from the Image Library for Intelligent Detection Systems (i-LIDS) [1]. i-LIDS is comprised of CCTV video surveillance videos consisting of 4 scenarios. Our focus is on the sterile zone monitoring scenario which consists of videos of a fence line under various situations and conditions: day/night, sun/rain, animals, people walking/running/rolling/carrying objects, etc. The data was organized into 2 classes: background and person which leads to a 2-class classification problem. A small detail in the description of the LSTM and LSM above, is that the temporal dimension is required to be the constant between inputs. The videos contained in the i-LIDS have varying lengths and therefore the data needs one more step of preprocessing. To solve this problem, we define a constant size window and slide it through the video, creating multiple constant length videos from a single video. Additionally, frames in the video may be subsampled to decrease the length of the video in order to aid in training of the algorithms.

TABLE I
Algorithm accuracy with varying preprocessing.

| | LSTM accuracy | LSM accuracy | Ensemble accuracy |
|---|---|---|---|
| Inception-v3 | .931 | .923 | .927 |
| Googlenet | .862 | .886 | .874 |

TABLE II
Algorithm accuracy with varying window size and sub sampling rates.

| Sub-sampling rate | Window size | LSTM accuracy | LSM accuracy | Ensemble accuracy |
|---|---|---|---|---|
| 1 | 50 | .929 | .922 | .9255 |
| 2 | 25 | .931 | .922 | .9265 |
| 5 | 10 | .931 | .923 | .927 |
| 10 | 5 | .908 | .915 | .9115 |
| 25 | 2 | .894 | .913 | .9035 |
| 50 | 1 | .880 | .906 | .893 |

### A. Experiment 1 - Algorithm performance

The first experiment tests the baseline performance of the algorithms on the i-LIDS data set described above. A subsampling rate of 5 frames and window size of 10 frames was used and the data set was preprocessed in 2 ways: features extraction from pretrained Inception-v3 CNN [12], and feature extraction from pretrained Googlenet CNN [11]. The results are shown in Table I. Both experiments show high performance, although Inception-v3 preprocessing out performed the Googlenet preprocessing.

### B. Experiment 2 - Subsampling rate and window size

The second experiment tests the affect on choosing the subsampling rate and window size on the i-LIDS data set. Based on the first experiment, we conduct this one using features from the Inception-v3 CNN. The parameters and results for the LSTM and LSM are shown in Table II. The results show the best performance when there is a balance between subsampling rate and window size. If the subsampling rate is too small then there is not as much differentiation between frames for the temporal algorithms to learn. On the other hand, if the window size is too small then we are not leveraging the memory benefits of using temporal algorithms to the full potential.

### C. Experiment 3 - Comparison of LSTM and LSM

The last experiment digs into the results of the LSTM and LSM individually and investigates how each algorithm performs on the i-LIDS data set. Although both algorithms perform fairly accurately, we want to investigate the areas where the algorithms are wrong. For physical security applications, a false positive results in a nuisance or false alarm while a false negative results in missing of an intrusion. Ideally, we want no false negatives and low false positives, but in reality we want to lower false positives and false negatives as much as possible.

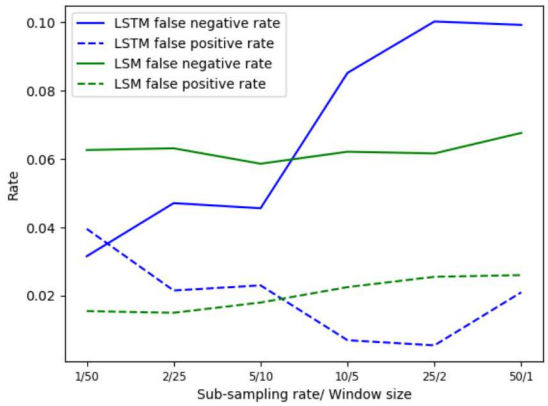


Fig. 7. Comparison of LSTM and LSM false positive and false negative rates

Fig. 7 shows the false positive and false negative rates for both the LSTM and LSM using the same parameters as Experiment 2. For smaller subsampling rates and larger window sizes, the LSM has a better false positive rate while the LSTM had a better false negative rate. Shortly after the subsampling rate/window size that gave the best performance, the LSTM has a better false positive rate while the LSM had a better false negative rate. These results show that the strengths of each algorithm depend on preprocessing which needs to be further explored in order to best combine the algorithms into an ensemble with both low false positive and false negative rates.

## V. Future Work

The results described in section IV only begin to investigate the benefits of using temporal machine learning to assist operators and reduce NAR/FAR. The results in this paper present good classification results and suggest that this approach should be further studied. There are many directions for future work to improve the architecture described in this paper.

First, we want to implement a learning ensemble. The ensemble will be similar to the one described in section III-B3, except that the weights of the voting will also be learned through an additional algorithm. In addition, we will explore inclusion of additional algorithms into this ensemble such as temporal frequency analysis (TFA) [13] and 3D convolutional neural networks [3] and determine the benefits each algorithm will provide to the ensemble.

Second, we want to explore hyperparameter optimization of each algorithm to improve performance. Each algorithm has different parameters (such as how many nodes or the activation functions within each node) that can be tuned to improve performance. Hyperparameter optimization seeks to find the combination of such parameters which yields the best performance. No hyperparameter optimization was done to obtain the results in section IV, so we conjecture that hyperparameter optimization will improve the results shown in this paper.

The results presented in this paper were not as good as the previous work performing frame-by-frame classification using transfer learning [10]. These previous results suggest that transfer learning is helpful in classifying the i-LIDS data set. The next step for future work is to use the resulting CNN from the transfer learning in [10] as a preprocessor for the temporal algorithms.

Once the above additions are incorporated into the architecture, we want to analyze its performance on higher fidelity data. This includes continuing to use i-LIDS, but performing a finer level of classification by distinguishing between walking, running, crawling, etc. We will also investigate how the architecture performs on data from different physical security sites with varying camera views.

Overall, we aim to continually improve the architecture described in this paper to develop an architecture that will aid alarm station operators in evaluating threats and allows them to focus on the most relevant threats. This will be accomplished by producing an architecture that has low false negative rates (i.e., low NAR/FAR rates), but also has a low false positive rate (i.e., don't miss an intrusion).

## References

[1] HOSD Branch. Imagery library for intelligent detection systems (i-lids). In *The Institution of Engineering and Technology Conference on Crime and Security*, pages 445–448, 2006.

[2] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

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

[5] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[6] Herbert Jaeger. The echo state approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[8] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[9] Judi E See. Vigilance: A review of the literature and applications to sentry duty. *Technical report*, SAND2014-17929, 2014.

[10] Jaclynn J Stubbs, Gabriel C Birch, Bryana L Woo, and Camron G Kouhestani. Physical security assessment with convolutional neural network transfer learning. In *Proceedings of the 2017 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2017.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[12] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[13] Bryana L Woo, Gabriel C Birch, Jaclynn J Stubbs, and Camron G Kouhestani. Unmanned aerial system detection and assessment through temporal frequency analysis. In *Proceedings of the 2017 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2017.