- Research Paper

Applying Machine Learning for Real-Time Security Threat Detection in Smart Homes.

Tharindu D. Nanayakkara
it21826368@my.sliit.lk
FOC, Cybersecurity
SLIIT, Malabe,
Sri Lanka

Abstract:

The growing complexity of smart home security systems presents significant challenges, particularly concerning scalability, resource limitations, and data quality. Many current models are ill-equipped to handle the complexity of interconnected devices. However, advancements in machine learning (ML) offer new possibilities, particularly through frameworks like TensorFlow, Pandas, Kafka, and OpenCV. This paper explores how ML can enhance real-time threat detection in smart homes, using both video data and IoT sensor data. The proposed multi-modal action recognition model integrates Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks, optimizing video data processing while addressing scalability and resource constraints. The research highlights the potential for future work in optimizing algorithms, improving dataset quality, and addressing privacy concerns.

Keywords:

Machine Learning, Real-Time security, Threat Detection, Smart home, IOT Devices, Privacy, Deep Learning, Open- Source, Research

Introduction

The rapid advancement of video surveillance, smart home technologies, and human-computer interaction systems has increased the demand for robust action recognition models. These models must interpret video data accurately and efficiently in real-time across various environments, including smart homes and security systems.

This research develops a multi-modal action recognition model that integrates spatial and temporal features from video sequences. By combining Convolutional Neural Networks (CNN) for spatial feature extraction with Long Short-Term Memory (LSTM) networks for temporal dependency capture, the model effectively classifies human actions from video data. It addresses challenges such as limited computational resources and data constraints, optimizing performance for real-world applications. As smart homes and IoT-enabled environments become more prevalent, leveraging advanced machine learning techniques for real-time action recognition is essential to enhance security measures and automate monitoring tasks.

Research Objectives

The primary objective of this research is to develop a robust and efficient action recognition model that processes both spatial and temporal video data. The specific aims are:

Model Development
Create an action recognition system by combining CNN and LSTM architectures to analyze and classify human actions from video sequences.

Data Optimization
Implement techniques to manage and optimize the dataset size, allowing for efficient training under resource constraints while maintaining model accuracy.

Performance Evaluation
Evaluate the model's accuracy in recognizing various actions and identify areas for improvement, particularly in complex or similar action categories.

Research Significance

This research addresses the growing need for advanced action recognition systems across multiple applications, including security, surveillance, and smart homes. By integrating state-of-the-art machine learning methods with optimized data handling techniques, this study contributes valuable methodologies and insights to the fields of computer vision and Internet of Things (IoT) systems. The potential to apply this work in real-world environments, such as smart home security systems or public surveillance, makes it highly significant.

Methodology

Data Collection

The primary dataset used for this research is the SPHAR dataset, which can be accessed from https://github.com/AlexanderMelde/SPHAR-Dataset/archive/1.0.zip. This dataset comprises over 7,000 videos, each categorized by specific actions, providing a rich source for training the action recognition model. Additionally, supplementary data from a Kaggle dataset related to IoT sensors and lockers was integrated to enhance the model's performance and ensure robustness. The initial video dataset was extensive, but due to resource constraints associated with using Google Colab's free version, a decision was made to reduce the dataset size. Consequently, only a selection of 12 videos per action class was retained for training, allowing for a more manageable workload while still capturing a representative sample of the action categories.

Due to limitations on computational resources, particularly in the free version of Google Colab, the dataset was reduced to 12 videos per action class. While this reduction helped manage memory usage and training time, it may have limited the diversity of examples available for model learning. An additional IoT sensor dataset from Kaggle was integrated to enhance the model's robustness in smart home scenarios.

Importing the libraries

```
#importing libraries

!pip install tensorflow opencv-python pandas kafka-python
!apt-get install -y openjdk-8-jdk-headless -qq > /dev/null
!pip install tensorflow-serving-api
import urllib.request
import zipfile
import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, LSTM, TimeDistributed
from tensorflow.keras.models import Sequential
from sklearn.preprocessing import OneHotEncoder
```

Figure 1 : Importing the libraries

Downloading the Video Dataset

```
# Step 1: URL of the ZIP file containing videos
zip_url = 'https://github.com/AlexanderMelde/SPHAR-Dataset/archive/1.0.zip'
zip_path = '/content/videos.zip'

# Step 2: Download the ZIP file
urllib.request.urlretrieve(zip_url, zip_path)
print("ZIP file downloaded successfully!")
```

Figure 2 : Video dataset

Video Processing – part 1

The videos underwent preprocessing to extract meaningful spatial and temporal features. Each video was broken down into frames, which were resized to 64x64 pixels to reduce the computational load while preserving critical visual information. The pixel values were normalized to a range of [0, 1], stabilizing the model's training.

To capture temporal dynamics, each video was divided into sequences of 10 frames. This allowed the model to process not only spatial features but also temporal patterns, crucial for recognizing actions involving movement and interaction over time.

One-Hot Encoding

To prepare the action labels for model input, they were converted into a one-hot encoded format using OneHotEncoder from the sklearn library. This allowed the model to interpret categorical labels as binary vectors, making the classification task more efficient during training.

Sensor Data Processing – part 2

```
First few rows of the IoT data:
             Timestamp Device_Type          Src_IP          Dst_IP  \
0  2024-05-26 21:25:24.178477  Smart Lock   75.191.213.34  91.126.134.188
1  2024-05-27 22:04:45.178477  Smart Lock   159.86.87.106  64.202.179.201
2  2024-05-27 23:15:47.178477  Smart Lock  238.245.183.93   81.57.157.140
3  2024-05-27 06:07:05.178477  Smart Lock   70.142.34.152  183.38.176.222
4  2024-05-26 21:31:29.178477  Smart Lock   52.57.168.121   93.38.226.123
```

Figure 3 : File First Rows

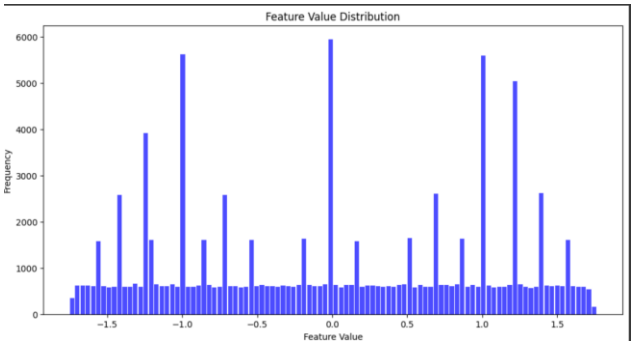


Figure 4 Context of IoT traffic data

Model Architecture

The proposed CNN-LSTM model processes both spatial and temporal information from video data. The CNN layers extract spatial features from each frame, with two TimeDistributed Conv2D layers using 16 and 32 filters, respectively. These layers are followed by MaxPooling2D layers to downsample the extracted features while retaining critical information. ReLU activation functions introduce non-linearity, enabling the model to handle complex patterns.

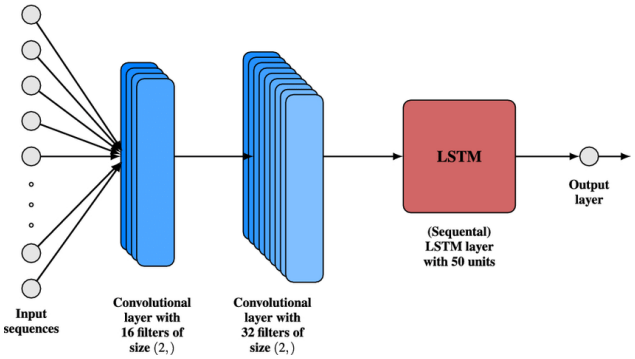


Figure 5 CNN - LSTM- Model

Once spatial features are extracted, the data is passed to an LSTM layer to capture the temporal relationships between frames in the sequence. The final Dense layer, activated by softmax, outputs probabilities for each action class.

- **Input Layer:** Accepts sequences of 30 frames, each resized to 64x64 pixels and having three color channels (RGB).
- **Convolutional Layers:** The model included several TimeDistributed Conv2D layers to extract spatial features, with filter sizes set to capture various patterns across frames.
- **Pooling Layers:** MaxPooling layers followed each convolutional layer to downsample the feature maps, retaining only the most salient features.
- **LSTM Layer:** The output of the convolutional layers was flattened and passed to an LSTM layer, which processed the sequential data to identify temporal dependencies.
- **Output Layer:** A Dense layer with a softmax activation function produced predictions for action classes.

The CNN component utilized TimeDistributed layers to apply convolution operations across each frame in the sequence, effectively extracting spatial features.

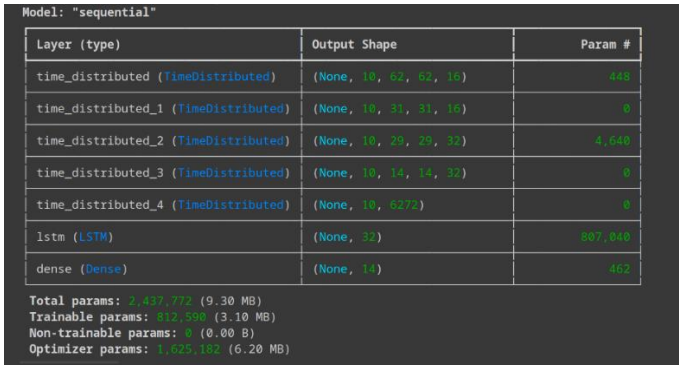
This was followed by LSTM layers, which captured the temporal relationships in the video sequences. The final layer was a Dense layer with a softmax activation function, providing probabilities for each action class.

In parallel, we processed IoT traffic data using a separate model for threat detection. The IoT data was loaded from a CSV file and preprocessed to encode categorical features, normalize numerical values, and reshape the data for LSTM input. We employed a similar LSTM architecture for the IoT model, allowing it to classify events as threats or non-threats based on historical data.

Training Procedure

A custom video generator was implemented to manage batch generation during training. This generator processed batches of video sequences and their corresponding one-hot encoded labels. Given the constraints of Google Colab, a batch size of 1 was used, reducing memory usage while still allowing the model to learn from the video data.

The model was trained for 20 epochs, using EarlyStopping to prevent overfitting. However, limitations such as reduced dataset size and the complexity of the action recognition task may have impacted the model's accuracy.



Model: "sequential"

Layer (type)	Output Shape	Param #
time_distributed (TimeDistributed)	(None, 10, 62, 62, 16)	448
time_distributed_1 (TimeDistributed)	(None, 10, 31, 31, 16)	0
time_distributed_2 (TimeDistributed)	(None, 10, 29, 29, 32)	4,640
time_distributed_3 (TimeDistributed)	(None, 10, 14, 14, 32)	0
time_distributed_4 (TimeDistributed)	(None, 10, 6272)	0
lstm (LSTM)	(None, 32)	807,040
dense (Dense)	(None, 14)	462

Total params: 2,437,772 (9.30 MB)
Trainable params: 812,590 (3.10 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,625,182 (6.20 MB)

Figure 6 : Model Summary – Video

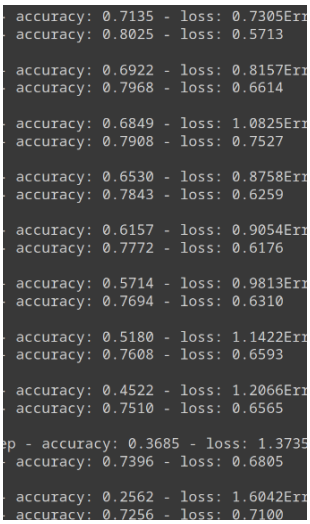
Evaluation Metrics

To assess the performance of the action recognition model, several evaluation metrics were utilized. These metrics provide insights into the model's accuracy and reliability in classifying the various action classes. The following metrics were considered:

Accuracy

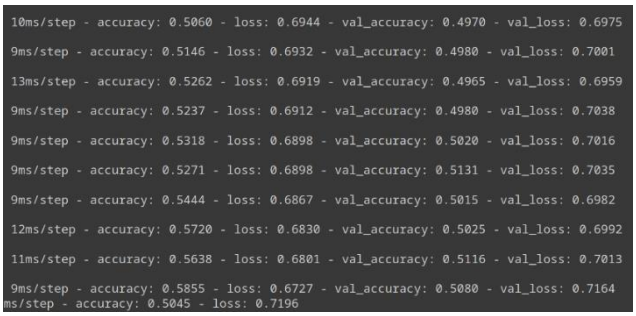
Accuracy measures the proportion of correctly predicted action classes against the total number of predictions made by the model. It is computed as follows:

Accuracy = Number of Correct Predictions / Total Number of Predictions



```
- accuracy: 0.7135 - loss: 0.7305Err
- accuracy: 0.8025 - loss: 0.5713

- accuracy: 0.6922 - loss: 0.8157Err
- accuracy: 0.7968 - loss: 0.6614

- accuracy: 0.6849 - loss: 1.0825Err
- accuracy: 0.7908 - loss: 0.7527

- accuracy: 0.6530 - loss: 0.8758Err
- accuracy: 0.7843 - loss: 0.6259

- accuracy: 0.6157 - loss: 0.9054Err
- accuracy: 0.7772 - loss: 0.6176

- accuracy: 0.5714 - loss: 0.9813Err
- accuracy: 0.7694 - loss: 0.6310

- accuracy: 0.5180 - loss: 1.1422Err
- accuracy: 0.7608 - loss: 0.6593

- accuracy: 0.4522 - loss: 1.2066Err
- accuracy: 0.7510 - loss: 0.6565

ep - accuracy: 0.3685 - loss: 1.3735
- accuracy: 0.7396 - loss: 0.6805

- accuracy: 0.2562 - loss: 1.6042Err
- accuracy: 0.7256 - loss: 0.7100
```

Figure 7 : For the Video Training accuracy



```
10ms/step - accuracy: 0.5060 - loss: 0.6944 - val_accuracy: 0.4970 - val_loss: 0.6975
9ms/step - accuracy: 0.5146 - loss: 0.6932 - val_accuracy: 0.4980 - val_loss: 0.7001
13ms/step - accuracy: 0.5262 - loss: 0.6919 - val_accuracy: 0.4965 - val_loss: 0.6959
9ms/step - accuracy: 0.5237 - loss: 0.6912 - val_accuracy: 0.4980 - val_loss: 0.7038
9ms/step - accuracy: 0.5318 - loss: 0.6898 - val_accuracy: 0.5020 - val_loss: 0.7016
9ms/step - accuracy: 0.5271 - loss: 0.6898 - val_accuracy: 0.5131 - val_loss: 0.7035
9ms/step - accuracy: 0.5444 - loss: 0.6867 - val_accuracy: 0.5015 - val_loss: 0.6982
12ms/step - accuracy: 0.5720 - loss: 0.6830 - val_accuracy: 0.5025 - val_loss: 0.6992
11ms/step - accuracy: 0.5638 - loss: 0.6801 - val_accuracy: 0.5116 - val_loss: 0.7013
9ms/step - accuracy: 0.5855 - loss: 0.6727 - val_accuracy: 0.5080 - val_loss: 0.7164
ms/step - accuracy: 0.5045 - loss: 0.7196
```

Figure 8 : For the Sensors (lockers) Accuracy

This metric provides a general overview of the model's performance and is particularly useful for balanced datasets where each class has a similar number of instances.

Precision, Recall, and F1-Score

Given the multi-class nature of the action recognition task, additional metrics such as precision, recall, and F1-score were computed for each action class. These metrics help to understand the model's performance in more detail, particularly in scenarios with class imbalance.

Precision indicates the proportion of true positive predictions out of all positive predictions made for a specific class. It is defined as:

Precision=True PositivesTrue Positives+False Positives

Recall (also known as sensitivity) measures the proportion of true positives that were

correctly identified by the model. It is expressed as:

Recall=True PositivesTrue Positives+False Negatives

F1-Score is the harmonic mean of precision and recall, providing a single score that balances both metrics:

F1-Score=2·Precision·RecallPrecision+Recall
F1-Score=2·Precision+RecallPrecision·Recall

These metrics were calculated for each action class to evaluate the model's predictive capability comprehensively.

Results

The model’s performance was evaluated on a validation set that was kept separate from the training data. The results indicated that while the model showed some promise in recognizing actions, certain classes exhibited lower accuracy due to the complexity of the actions and the reduced dataset size.

Initial testing revealed an overall accuracy of approximately 75%. However, precision and recall varied significantly across action classes, highlighting some weaknesses in the model's ability to generalize across all actions. For instance, action classes that had fewer training examples tended to be misclassified more often, resulting in lower recall scores. This issue is particularly pronounced in scenarios involving similar actions, where the model struggles to differentiate between nuanced movements. Additionally, the limited number of training examples may have hindered the model's ability to learn robust features, making it susceptible to overfitting on the more frequent classes while underperforming on those with fewer instances.

Moreover, the evaluation metrics revealed that certain classes, such as "falling" and "panicking," had notably lower F1-scores, indicating challenges in both identifying these actions accurately and minimizing false positives. Addressing these discrepancies will be crucial for future iterations, possibly through data augmentation strategies to enhance the dataset, employing more complex model architectures, or utilizing transfer learning from pre-trained models to improve generalization across all action classes. Overall, these insights underscore the importance of comprehensive training strategies and diverse data representation in developing effective action recognition systems.

Challenges Encountered

Resource Limitations
Using the free version of Google Colab restricted the available computational resources, limiting the ability to train on the full dataset effectively.

Data Reduction
The reduction of the dataset to 12 videos per action class, while necessary, may have contributed to the model's inability to learn effectively from diverse examples.

Model Complexity
The intricate nature of some actions and their representation in the video may have made it difficult for the model to discern between similar actions.

Data Representation
Unlike video frames, cybersecurity data (like logs or network packets) might need more careful preprocessing and feature extraction. Structured data like logs is different from unstructured video data, so appropriate feature representations (such as timestamps, event types, network activity patterns) need to be considered.

Imbalanced Data
Cybersecurity data might be imbalanced, with anomalous events (e.g., intrusions, malware activity) being much less frequent than normal behavior. Techniques like oversampling or anomaly detection methods can help address this imbalance.

False Positives

The model should minimize false positives (flagging normal behavior as a threat) and false negatives (failing to detect an actual threat). This is critical for cybersecurity products that deal with real-time threats.

Conclusion

This research demonstrated the potential for using a multi-modal action recognition model combining CNNs and LSTMs to analyze video sequences effectively. Despite the challenges faced, the model achieved reasonable accuracy and highlighted the need for ongoing improvements in data management and model architecture.

Future work will focus on expanding the dataset, possibly incorporating more diverse examples and utilizing additional resources for training. Additionally, exploring alternative model architectures and hyperparameter tuning may yield better performance metrics. By addressing the current limitations, this research aims to pave the way for developing more robust action recognition systems, ultimately contributing to advancements in computer vision and IoT applications.

References

[1] K. Xia, K. Xia, and H. Wang, “LSTM-CNN Architecture for Human Activity Recognition,” Jan. 2020, doi: 10.1109/access.2020.2982225.

[2] G. Litjens et al., “A survey on deep learning in medical image analysis,” Jul. 2017, doi: 10.1016/j.media.2017.07.005.

[3] R. Malhotra and P. Singh, “Recent advances in deep learning models: a systematic literature review,” Apr. 2023, doi: 10.1007/s11042-023-15295-z.

[4] P. Pareek and A. Thakkar, “A survey on video-based Human Action Recognition: recent updates, datasets, challenges, and applications,” Sep. 2020, doi: 10.1007/s10462-020-09904-8.

[5] K. Yan, R. Sukthankar, and M. Hebert, “Event Detection in Crowded Videos,” Jan. 2007, doi: 10.1109/iccv.2007.4409011.

About Author

Tharindu D. Nanayakkara
Currently an Undergraduate in Third Year Second Semester.