Wachemo University
College of Engineering and Technology
Department of Software Engineering

Project Title : <u>Smart/AI Security</u>

Names of project developer

1. Natinael Samuel.......................1366
2. Yitagesu Basazin.....................2218
3. Mustefa Gemechu...................1305204
4. Kerenso Haji...........................1305749
5. Tamirat Teshome.....................1409545
6. Kirubel Deke ………………...1402409

Submitted to: - Ms. Senedu G/mariam

11 November 15, 2023
Hossana, Ethiopia

Table Content

1. Introduction

Smart/AI security refers to the application of machine learning and other advanced technologies to enhance security measures and protect against various threats. Smart/AI security solutions aim to overcome security limitations by using machine learning algorithms and computer vision to provide robust security measures.

The primary advantage of smart/AI security is its ability to analyze vast amounts of data in real-time and identify patterns, anomalies, and potential threats that may go unnoticed by human operators. This real-time data analysis is particularly useful for monitoring and interpreting data from surveillance cameras. By analyzing this data, we can detect suspicious activities and potential vulnerabilities, such as unauthorized events or unusual behavior.

In conclusion, smart/AI security represents a significant advancement in safeguarding individuals, organizations, and critical infrastructure. By harnessing the power of machine learning, this systems can provide real-time threat detection and proactive security measures, helping to protect us from a wide range of threats.

2. Statement of the Problem

- From our interview during last semester for Requirement Engineering Assignment with the ICT developmental directorate of
- Wachemo University has implemented a CCTV camera system with real-time data storage, but its utilization is significantly lower than its intended capabilities. The current system only records footage, and security personnel only review it after receiving a report from the security office, which can take up to 21 days. This delayed response time hinders the university's ability to effectively prevent and respond to security incidents.
- To address this issue, we propose a new machine learning project that will automate and enhance violence detection using the existing CCTV infrastructure. Our project aims to:
  - ✓ Real-time violence detection: The system will analyze CCTV footage in real-time and immediately alert security personnel when violence is detected. This will enable a more proactive approach to security, allowing for faster response times and preventing incidents from escalating.
  - ✓ Behavior analysis of scenes: The system will analyze the behavior of individuals and groups captured by CCTV cameras. This will enable the identification of suspicious behavior patterns that may indicate potential threats, allowing security personnel to take preventive measures.

- By implementing these enhancements, we aim to significantly improve the effectiveness of Wachemo University's CCTV system, ensuring the safety and security of its students, staff, and visitors.

3. Objective of the project

- ➢ The general objective of project **Smart/AI Security** is to develop and implement a machine learning-powered CCTV surveillance system that enables real-time violence detection and behavior analysis to enhance security and safety at Wachemo University. Throuhg

    - ✓ Reducing security threats in risky rooms like server rooms, president office, students dormitory, warehouses, students laboratory, smart rooms, residential buildings inside the campus and so on.
    - ✓ Enhancing respond of security office in urgent security issues by analyzing threats and situations using Machine Learning algorithm by reducing the average response time to security incidents from 21 days to 1 minute.

4. Methodology of the Project

4. 1. Data collection

- ➢ The dataset for this project was meticulously curated from YouTube videos, specifically targeting those containing instances of physical altercations. Additionally, non-fight sequences were extracted from regular surveillance camera videos to ensure a comprehensive dataset. The organized dataset was obtained by cloning the repository from the IPTA 2019 conference warehouse. In total, there are 300 videos comprising 150 fight instances and 150 non-fight sequences. Each video is standardized to a length of 2 seconds.
- ➢ We have used various data collection methods to ensure diversity and relevance in the dataset:
    - ✓ **Web Scraping:** Extracting data from YouTube.
    - ✓ **Publicly Available and per-prosesd Datasets:** Using existing datasets of UCF-ARG (University of Central Florida) which is prepared for research and development purposes.

- ✓ **Collaboration with Institutions:** Accessing specialized datasets that are not publicly available from IPTA 2019 conference warehouse by establishing clear agreements and comply with ethical standards.

4. 2. Data analysis (pre-processing)

- ➢ In our project, we performed some pre-processing on the dataset. Firstly, we read the video files from the dataset and resized the frames of the videos to a fixed width and height. This was done to reduce computations. We normalized the data by dividing the pixel values by 255, facilitating faster convergence during the training of the network. The steps followed during data pre-processing were as follows:
  - Specify the height and width to which each video frame will be resized in our dataset.

    # IMAGE_HEIGHT , IMAGE_WIDTH = 64, 64
  - Specify the number of frames of a video that will be fed to the model as one sequence.

    # SEQUENCE_LENGTH = 20
  - Specify the directory containing the UCF50 dataset.

    # DATASET_DIR = "dataset/"
  - Specify the list containing the names of the classes used for training.

    #CLASSES_LIST = […]
- ➢ The constants IMAGE_HEIGHT, IMAGE_WIDTH, and SEQUENCE_LENGTH can be increased for better results. However, it's important to note that increasing the sequence length is effective only up to a certain point. Beyond this, increasing the values will make the process more computationally expensive. The next step of the project involved creating a function, in this case, called frames_extraction(). This function creates a list containing the resized and normalized frames of a video whose path is passed to it as an argument. The function reads the video file frame by frame, with only an evenly distributed sequence length of frames being added to the list.

- ✓ Args: video_path: The path of the video in the disk, whose frames are to be extracted.
- ✓ Returns: frames_list: A list containing the resized and normalized frames of the video.

- ➢ Subsequently, another function called create_dataset() was created. This function extracts the data of the selected classes and creates the required dataset. It returns features, labels, and video files paths.
    - ✓ features: A list containing the extracted frames of the videos.
    - ✓ labels: A list containing the indexes of the classes associated with the videos.
    - ✓ video_files_paths: A list containing the paths of the videos in the disk.

In the above function, the following tasks were performed during data pre-processing:

- Declared Empty Lists to store the features, labels and video file path values.
- Iterating through all the classes mentioned in the classes list
- Display the name of the class whose data is being extracted.
- Get the list of video files present in the specific class name directory.
- Iterate through all the files present in the files list.
- Get the complete video path.
- Extract the frames of the video file.
- Check if the extracted frames are equal to the SEQUENCE_LENGTH specified above. So ignore the vides having frames less than the SEQUENCE_LENGTH.
- Append the data to their repective lists.
- Convert the list to numpy arrays
- Return the frames, class index, and video file path.
-

4. 3. Model Selection

Model Information

Model Name: [LRCN and ConvLSTM]

Framework: [TensorFlow, PyTorch, opencv, moviepy, pydot]

Architecture: [CNN-LSTM]

Training Dataset: [75%]

Validation Dataset: [External]

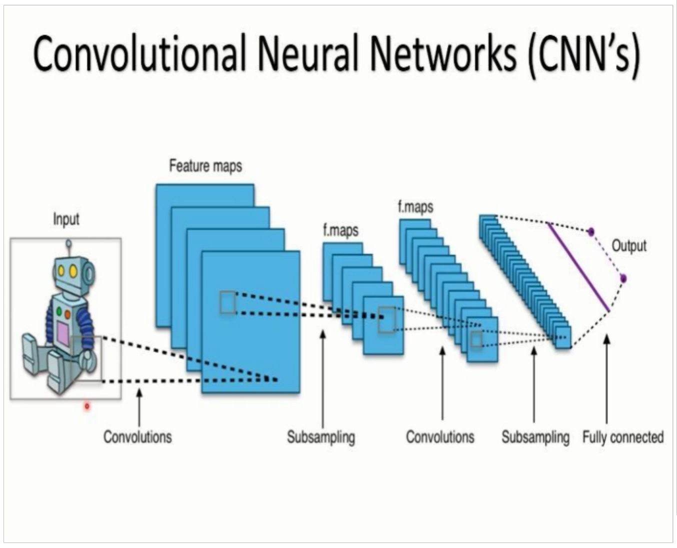
Testing Dataset: [25%]

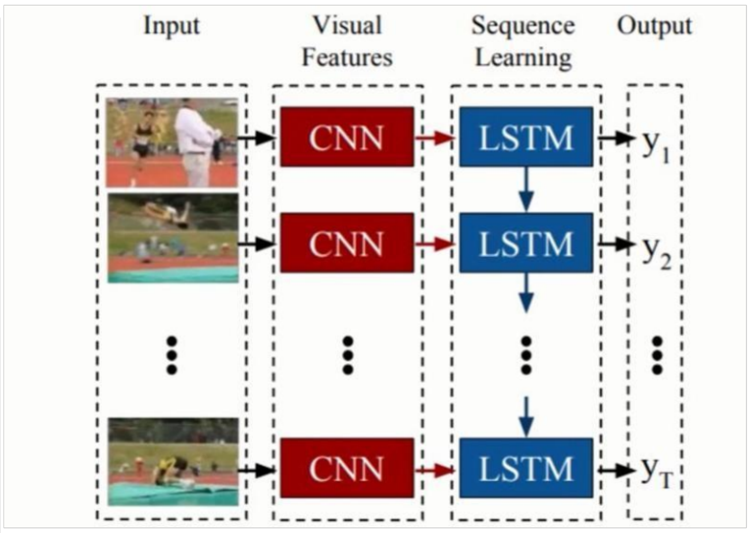
- In this Project to we will use CNN and LSTM approach . CNNs and LSTMs are two of the most powerful machine learning models for processing sequential data. CNNs are well-suited for extracting spatial features from data, while LSTMs are well-suited for capturing temporal dependencies. This makes them a powerful combination for tasks such as video prediction, image sequence generation, and natural language processing.
- The CNN-LSTM architecture approach in machine learning represented a hybrid approach that combined the strengths of convolutional neural networks (CNNs) and long short-term memory networks (LSTMs) to learn spatio-temporal dependencies in data. CNNs excelled at extracting spatial features, while LSTMs were effective at learning sequential dependencies. Combining these networks allowed the CNN-LSTM approach to predict the next state of a sequence based on the current state and previous states, considering spatial relationships between sequence elements.
- **On this project** we have used CNN-LSTM architecture using two different models variants. The two variants under consideration are the Long-term Recurrent Convolutional Network (LRCN) and the Convolutional Long Short-Term Memory (ConvLSTM).
    - ✓ **LRCN Model (Long-term Recurrent Convolutional Network)**
        - Overview: Combines Convolutional Neural Networks (CNNs) for spatial feature extraction with Long Short-Term Memory (LSTM) networks for sequential modeling.
        - Considerations: Potential computational complexity due to sequential layer application.

- ✓ **ConvLSTM Model (Convolutional Long Short-Term Memory)**
  - Overview: Integrates convolutional operations directly into LSTM cells, enabling joint spatial and temporal processing.
  - Considerations: Potential efficiency gains due to integrated operations.

The Criteria we used to select the architecture and model variants are

- Task Requirements : Assess the specific requirements of the action recognition task.
- Computational Efficiency : hardware and resource constraints for training and deployment.
- Flexibility : adaptability to potential changes in task requirements.
- Performance Metrics: Accuracy and validation performance on datasets.





4. 4. Performance measurement

After training, we will evaluate the model on the test set. The performance of the model will be measured using a variety of metrics, including

- Total Accuracy and Total Validation Accuracy
- Total Loss and Total Validation Loss.
- Confusion Matrix
- Execution Time and Resource Usage

5. Model analysis

Analyzing the Convlstm Model layers and parameters:

```
 Layer (type)                          Output Shape                  Param #
=================================================================
conv_lstm2d (ConvLSTM2D)               (None, 20, 62, 62, 4)          1024
max_pooling3d (MaxPooling3D)           (None, 20, 31, 31, 4)          0
time_distributed (TimeDistributed)     (None, 20, 31, 31, 4)          0
conv_lstm2d_1 (ConvLSTM2D)             (None, 20, 29, 29, 8)          3488
max_pooling3d_1 (MaxPooling3D)         (None, 20, 15, 15, 8)          0
time_distributed_1 (TimeDistributed)   (None, 20, 15, 15, 8)          0
conv_lstm2d_2 (ConvLSTM2D)             (None, 20, 13, 13, 14)         11144
max_pooling3d_2 (MaxPooling3D)         (None, 20, 7, 7, 14)           0

Total params: 38762 (151.41 KB)
 Trainable params: 38762 (151.41 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Analyzing the LRCN Model layers and parameters:

```
 Layer (type)                             Output Shape                  Param #
=================================================================
time_distributed_3 (TimeDistributed)  (None, 20, 64, 64, 16)          448
time_distributed_4 (TimeDistributed)  (None, 20, 16, 16, 16)            0
time_distributed_5 (TimeDistributed)  (None, 20, 16, 16, 16)            0
time_distributed_6 (TimeDistributed)  (None, 20, 16, 16, 32)          4640
time_distributed_7 (TimeDistributed)  (None, 20, 4, 4, 32)              0
time_distributed_8 (TimeDistributed)  (None, 20, 4, 4, 32)              0
time_distributed_9 (TimeDistributed)  (None, 20, 4, 4, 64)            18496

Total params: 72994 (285.13 KB)
Trainable params: 72994 (285.13 KB)
 Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

6. Result

Compiling & Training the Convlstm Model

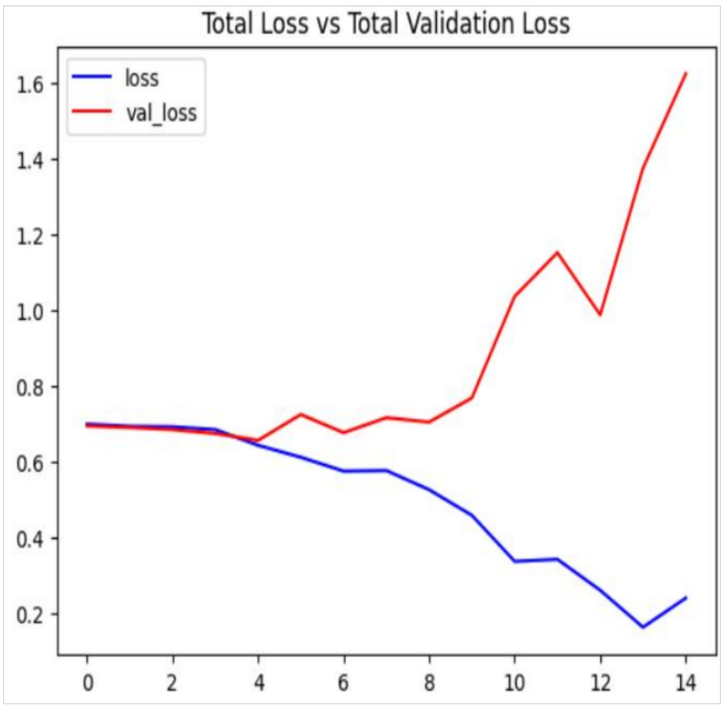
```
Epoch 1/5045/45 [==============================] - 104s 2s/step - loss: 0.6982 -
accuracy: 0.4778 - val_loss: 0.6933 - val_accuracy: 0.4889
Epoch 2/5045/45 [==============================] - 87s 2s/step - loss: 0.6923 -
accuracy: 0.4833 - val_loss: 0.6891 - val_accuracy: 0.4667Epoch 3/5045/45
[==============================] - 83s 2s/step - loss: 0.6911 - accuracy: 0.5944 -
val_loss: 0.6836 - val_accuracy: 0.5333
Epoch 4/5045/45 [==============================] - 99s 2s/step - loss: 0.6837 -
accuracy: 0.5556 - val_loss: 0.6733 - val_accuracy: 0.5556Epoch 5/5045/45
[==============================] - 90s 2s/step - loss: 0.6422 - accuracy: 0.6333 -
val_loss: 0.6553 - val_accuracy: 0.5556
Epoch 6/5045/45 [==============================] - 82s 2s/step - loss: 0.6105 -
accuracy: 0.7000 - val_loss: 0.7236 - val_accuracy: 0.5111
```

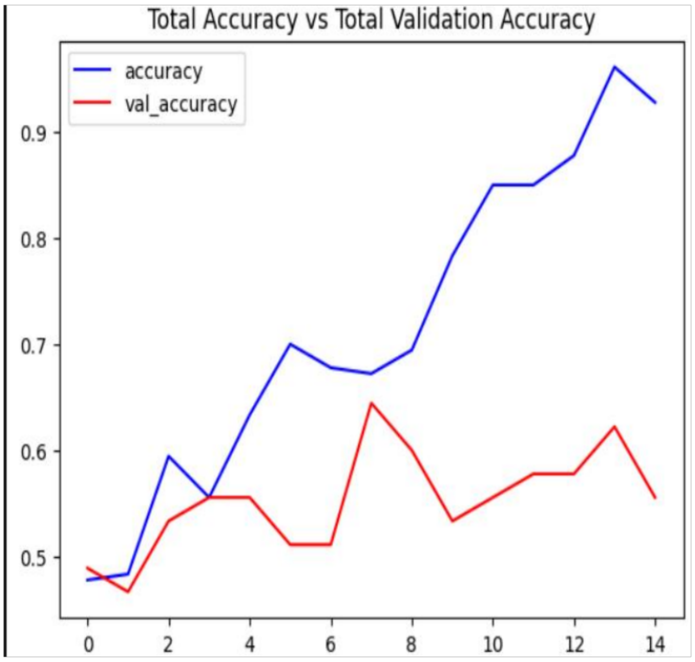
```
Epoch 7/5045/45 [==============================] - 90s 2s/step - loss: 0.5739 -
accuracy: 0.6778 - val_loss: 0.6754 - val_accuracy: 0.5111
Epoch 8/5045/45 [==============================] - 84s 2s/step - loss: 0.5754 -
accuracy: 0.6722 - val_loss: 0.7151 - val_accuracy: 0.6444
Epoch 9/5045/45 [==============================] - 90s 2s/step - loss: 0.5248 -
accuracy: 0.6944 - val_loss: 0.7035 - val_accuracy: 0.6000Epoch 10/5045/45
[==============================] - 85s 2s/step - loss: 0.4571 - accuracy: 0.7833 -
val_loss: 0.7672 - val_accuracy: 0.5333
Epoch 11/5045/45 [==============================] - 93s 2s/step - loss: 0.3354 -
accuracy: 0.8500 - val_loss: 1.0356 - val_accuracy: 0.5556
Epoch 12/5045/45 [==============================] - 89s 2s/step - loss: 0.3408 -
accuracy: 0.8500 - val_loss: 1.1513 - val_accuracy: 0.5778Epoch 13/50
45/45 [==============================] - 83s 2s/step - loss: 0.2592 - accuracy:
0.8778 - val_loss: 0.9869 - val_accuracy: 0.5778
Epoch 14/50 45/45 [==============================] - 91s 2s/step - loss: 0.1613 -
accuracy: 0.9611 - val_loss: 1.3746 - val_accuracy: 0.6222
Epoch 15/50 45/45 [==============================] - 92s 2s/step - loss: 0.2384 -
accuracy: 0.9278 - val_loss: 1.6238 - val_accuracy: 0.5556
```

Evaluating the Trained Model on the test set

```
3/3 [==============================] - 7s 2s/step - loss: 0.7096 - accuracy: 0.4667
```

Plotting Model's Loss & Accuracy Curves to justify weather this model is well fitted to be used in our project





Compiling & Training the LRCN Model

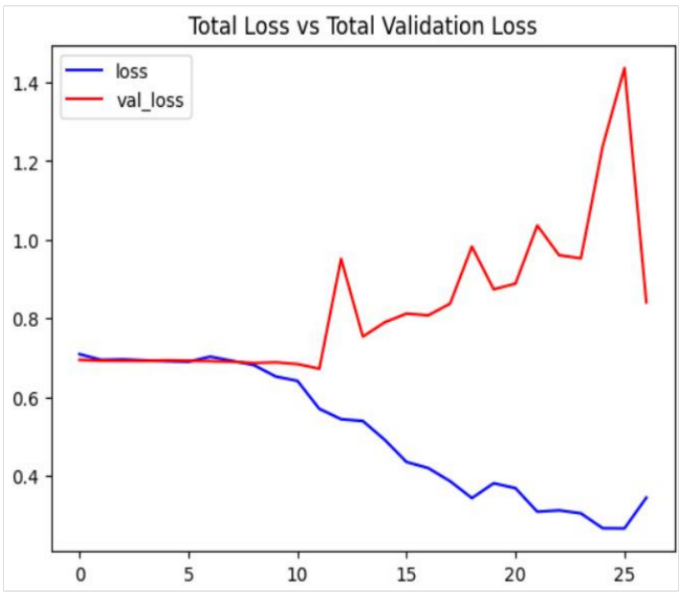
```
Epoch 1/7045/45 [==============================] - 25s 338ms/step - loss: 0.7094
- accuracy: 0.4611 - val_loss: 0.6949 - val_accuracy: 0.4889
Epoch 2/7045/45 [==============================] - 9s 209ms/step - loss: 0.6947 -
accuracy: 0.4833 - val_loss: 0.6927 - val_accuracy: 0.5333
Epoch 3/7045/45 [==============================] - 13s 287ms/step - loss: 0.6960
- accuracy: 0.5056 - val_loss: 0.6924 - val_accuracy: 0.5111
Epoch 4/7045/45 [==============================] - 10s 216ms/step - loss: 0.6934
- accuracy: 0.5000 - val_loss: 0.6922 - val_accuracy: 0.5778Epoch 5/7045/45
[==============================] - 11s 245ms/step - loss: 0.6915 - accuracy:
0.5000 - val_loss: 0.6935 - val_accuracy: 0.4889
Epoch 6/7045/45 [==============================] - 9s 192ms/step - loss: 0.6900 -
accuracy: 0.5278 - val_loss: 0.6925 - val_accuracy: 0.5333
Epoch 7/7045/45 [==============================] - 11s 245ms/step - loss: 0.7030
- accuracy: 0.5222 - val_loss: 0.6915 - val_accuracy: 0.5111
```

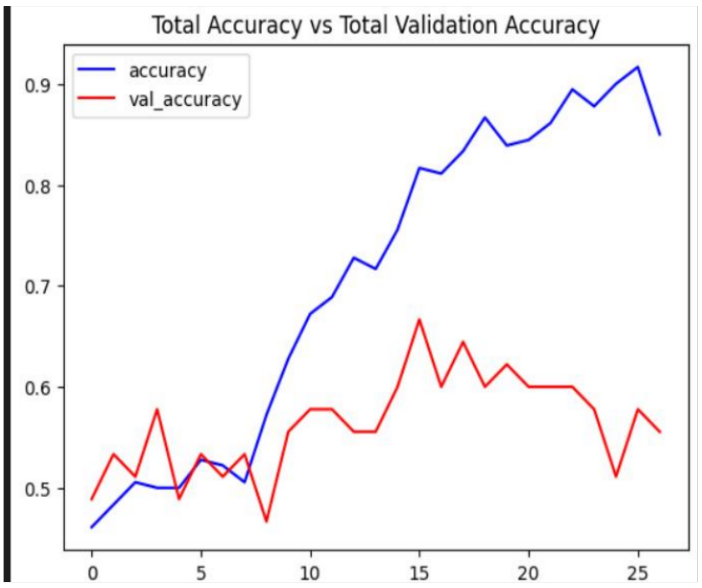
```
Epoch 8/7045/45 [==============================] - 8s 186ms/step - loss: 0.6922 -
accuracy: 0.5056 - val_loss: 0.6902 - val_accuracy: 0.5333
Epoch 9/7045/45 [==============================] - 15s 323ms/step - loss: 0.6815
- accuracy: 0.5722 - val_loss: 0.6868 - val_accuracy: 0.4667
Epoch 10/7045/45 [==============================] - 11s 241ms/step - loss:
0.6526 - accuracy: 0.6278 - val_loss: 0.6885 - val_accuracy: 0.5556
Epoch 11/7045/45 [==============================] - 11s 242ms/step - loss:
0.6413 - accuracy: 0.6722 - val_loss: 0.6840 - val_accuracy: 0.5778
Epoch 12/7045/45 [==============================] - 8s 185ms/step - loss: 0.5704
- accuracy: 0.6889 - val_loss: 0.6722 - val_accuracy: 0.5778
Epoch 13/70
45/45 [==============================] - 11s 254ms/step - loss: 0.2669 - accuracy:
0.9000 - val_loss: 1.2376 - val_accuracy: 0.5111
 Epoch 26/70 45/45 [==============================] - 7s 165ms/step - loss:
0.2665 - accuracy: 0.9167 - val_loss: 1.4369 - val_accuracy: 0.5778
Epoch 27/70 45/45 [==============================] - 11s 250ms/step - loss:
0.3445 - accuracy: 0.8500 - val_loss: 0.8411 - val_accuracy: 0.5556
```

Evaluating the Trained Model on the test set

```
3/3 [==============================] - 1s 297ms/step - loss: 0.7313 - accuracy:
0.4933
```

Plotting Model’s Loss & Accuracy Curves to justify weather this model is well fitted to be used in our project





7. Anlyzing Result and Decision

1. ConvLSTM Model:

Training Results:

- ✓ Epochs 1-5: The model demonstrated a gradual improvement in accuracy, reaching 63.33%. However, fluctuations in validation accuracy were observed, indicating possible overfitting.
- ✓ Epochs 6-10: Despite achieving 70% accuracy, there was an increase in validation loss, suggesting the model might struggle with generalization.
- ✓ Epochs 11-15: The model showed signs of overfitting, as training accuracy increased significantly while validation accuracy plateaued.
- ✓

Testing Results:

- ✓ Accuracy: 46.67%
- ✓ Observation: The model's performance on the test set was suboptimal, possibly due to overfitting during training.

2. **LRCN Model:**

Training Results:

- ✓ Epochs 1-5: The model demonstrated stability in accuracy, reaching 50%. Validation accuracy showed modest improvement.
- ✓ Epochs 6-10: The model continued to improve, reaching 67.22% accuracy, indicating a balanced learning process.
- ✓ Epochs 11-15: Training accuracy reached 68.89%, but there were fluctuations in validation accuracy, indicating a potential need for regularization.

Testing Results:

- ✓ Accuracy: 49.33%
- ✓ Observation: The LRCN model performed marginally better than the ConvLSTM model on the test set.

Decision on Model Selection:

➢ The LRCN model exhibited more stable training and validation accuracy over epochs. Despite both models facing challenges, the LRCN model's marginally better performance on the test set suggests it might generalize more effectively.

Unexpected Outcomes:

- ➢ ConvLSTM Model: Training accuracy significantly outpaced validation accuracy, indicating potential overfitting. Fluctuations in validation accuracy suggested difficulties in generalizing to unseen data.
- ➢ LRCN Model: Validation accuracy showed fluctuations despite stable training accuracy.

Testing accuracy did not significantly surpass the ConvLSTM model, suggesting room for improvement.

8. Conclusion

- ➢ The ConvLSTM model exhibited signs of overfitting during training, with a significant gap between training and validation accuracy. Fluctuations in validation accuracy indicated difficulties in generalizing to unseen data. However, the model's testing accuracy of 46.67% suggests limited effectiveness on the test set. The ConvLSTM model may benefit from regularization techniques and adjustments to improve generalization.

- The LRCN model demonstrated a more stable training and validation accuracy progression, reaching 67.22% accuracy. Despite fluctuations in validation accuracy, the LRCN model outperformed the ConvLSTM model on the test set with an accuracy of 49.33%. This suggests that, although there is room for improvement, the LRCN model has a marginally better ability to generalize to new, unseen data.

9. Recommendations for Advanced Improvement

Both Models:

- ✓ **Regularization Techniques**: Implementing dropout layers or other regularization methods to mitigate overfitting, especially for the ConvLSTM model.
- ✓ **Hyperparameter Tuning**: Fine-tuning hyperparameters, such as learning rates and batch sizes, to optimize model convergence and performance.
- ✓ **Data Augmentation**: Exploring additional data augmentation strategies to increase the diversity of the training set and enhance model generalization.

ConvLSTM Model:

- ✓ **Architecture Adjustments:** Adjusting the model architecture to capture more complex spatiotemporal features in the data.
- ✓ **Training Strategy:** Experimenting with different training strategies, such as adjusting the sequence length or incorporating attention mechanisms

LRCN Model:

- ✓ **Validation Stability:** Investigating the source of fluctuations in validation accuracy and considering model adjustments or additional regularization.
- ✓ **Model Complexity:** Explore whether increasing the complexity of the LRCN model, such as adding more layers or units, can improve its ability to learn intricate patterns.

10. References

TensorFlow Documentation:

- TensorFlow. (n.d.). TensorFlow Documentation. Retrieved from https://www.tensorflow.org/api_docs

Keras Documentation:

- Chollet, F. (2015). Keras: The Python Deep Learning library. Retrieved from https://keras.io/

SciKit-Learn Documentation:

- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830. Retrieved from https://scikit-learn.org/stable/documentation.html

Python Documentation:

- Python Software Foundation. (n.d.). Python 3.9.7 Documentation. Retrieved from https://docs.python.org/3/

ConvLSTM Layer in Keras:

- Keras. (n.d.). ConvLSTM2D layer. Retrieved from https://keras.io/api/layers/recurrent_layers/conv_lstm2d/

LSTM Layer in Keras:

- Keras. (n.d.). LSTM layer. Retrieved from https://keras.io/api/layers/recurrent_layers/lstm/

Machine Learning Metrics:

- Scikit-Learn. (n.d.). Metrics and Scoring: quantifying the quality of predictions. Retrieved from https://scikit-learn.org/stable/modules/model_evaluation.html

Machine Learning Best Practices:

- Raschka, S., & Mirjalili, V. (2019). Python Machine Learning (3rd ed.). Packt Publishing.

Deep Learning Best Practices:

- Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep Learning (Vol. 1). MIT press Cambridge.

- Machine Learning Model Evaluation:
- Brownlee, J. (2020). Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/start-here/#algorithms