



Ain Shams University  
Faculty of Computer & Information Sciences  
Department of Computer Science

# VigilantEye: Violence detection

This documentation is submitted as required for the degree of bachelor's in computer and information sciences

By

Mohammed Nasr Abd-Elazem	Computer Science
Hesham Ayman Mohammad	Computer Science
Sama Alaa El-Din Hegazy	Computer Science
Heba Mohammad Saleh	Computer Science
Maryam Ahmed Abdullah	Computer Science

## **Under Supervision of**

Prof. Dr. Abeer Mahmoud  
Head of Computer Science Department,  
Faculty of Computer and Information Sciences,  
Ain Shams University.

T.A. Mirna Al-Shetairy  
Teaching Assistant,  
Computer Science Department,  
Faculty of Computer and Information Sciences,  
Ain Shams University.

**June 2024**

## **Acknowledgment**

We would like to use this opportunity to express our gratitude to everyone who supported us throughout this project. We are thankful for their indispensable guidance, invaluable constructive criticism, friendly advice, and the most obliged provision of their genuine and illuminating views.

We would like to offer our special thanks to our supervisor Prof. Dr. Abeer Mahmoud for her help and support.

We would also like to extend our thanks to T.A. Mirna Al-Shetairy for guiding and helping us throughout the whole course of the project; her help was invaluable to us.

## Abstract

Due to the increasing worries about the general public's safety, many surveillance cameras have been installed over the years to increase security. As video surveillance has become more widespread, the amount of video data has become too immense to be tracked by humans efficiently. Additionally, in case video evidence is found by the authorities for previous crimes, a great amount of effort is exerted until the very vital few seconds that have the vital information are found, therefore making there a need for a system able to automatically detect violence. Computer vision algorithms have been getting increasingly effective at object detection- including moving objects, as well as human action recognition. In this project, we developed VigilantEye, a user-friendly application that enables users to upload videos or start a livestream, for the purpose of being able to determine whether violent actions are being committed in said video or not. Our work focused on achieving two main goals: building a model that detects violent actions in videos efficiently, and another that clusters similar types of violence seen in our dataset together, using unsupervised learning, to further classify the type of violence detected in videos. The proposed methods are tested on a publicly available RWF-2000 dataset, on which different deep learning algorithms were applied. Our solution achieves an accuracy of 89.25% using a Sep-ConvLSTM model. As for the clustering problem, we found that DBSCAN is best fitting as it was successful in grouping the data into 2 groups.

بسبب القلق المتزايد على سلامة عامة الناس، تم تركيب العديد من كاميرات المراقبة على مدار السنوات لزيادة الأمن، ومع انتشار كاميرات المراقبة، أصبح حجم بيانات الفيديو هائلاً لدرجة أنه من الصعب على البشر متابعتها بكفاءة. بالإضافة إلى ذلك، عندما تجد السلطات أدلة فيديو لجرائم سابقة، يُبذل جهد كبير للعثور على الثواني الحيوية التي تحتوي على المعلومات المهمة، مما يبرز الحاجة إلى نظام قادر على اكتشاف العنف تلقائياً، ونظراً لأن خوارزميات الرؤية الحاسوبية أصبحت فعالة بشكل متزايد في اكتشاف الأشياء، بما في ذلك الأشياء المتحركة، وكذلك في التعرف على تصرفات البشر. لذلك قمنا في هذا المشروع بتطوير VigilantEye، وهو تطبيق سهل الاستخدام يتيح للمستخدمين تحميل الفيديوهات أو بدء بث مباشر، بهدف تحديد ما إذا كانت هناك أفعال عنيفة تُرتكب في الفيديو أم لا، ولقد تم تركيز عملنا على تحقيق هدفين رئيسيين: بناء نموذج يكتشف الأفعال العنيفة في الفيديوهات بكفاءة، وآخر يقوم بتجميع أنواع العنف المشابهة في مجموعات معاً، لتصنيف نوع العنف المكتشف في الفيديوهات. تم اختبار الطرق المقترحة على مجموعة بيانات RWF-2000 المتاحة للعامة، حيث تم تطبيق خوارزميات التعلم العميق المختلفة. حققنا دقة بلغت 89.25% باستخدام نموذج SepConvLstm-C. أما بالنسبة للجزء الآخر وهو جمع أنواع العنف المتشابهة في مجموعات، فقد وجدنا أن DBSCAN هو الأكثر ملاءمة لأنه نجح في تقسيم البيانات إلى مجموعتين.

# Table of Contents

Acknowledgment.....	ii
Abstract.....	iii
List of Figures.....	vii
List of Tables.....	viii
List of Abbreviations.....	ix
Chapter 1. Introduction.....	1
1.1 Problem Definition.....	1
1.2 Motivation.....	1
1.3 Objectives.....	1
1.4 Methodology.....	2
1.5 Time Plan.....	2
1.6 Thesis Outline.....	2
Chapter 2. Background.....	4
2.1 Literature Review.....	4
2.2 Techniques.....	6
2.2.1 ConvLSTM.....	6
2.2.2 C3D.....	10
2.2.4 DenseNet-121.....	14
2.2.5 Transformer.....	15
Chapter 3. System Architecture.....	17
3.1 System Overview.....	17
3.1.1 System Architecture.....	17
3.1.2 System Users.....	18
3.2 System Analysis & Design.....	19
3.2.1 Use Case Diagram.....	19
3.2.2: Flow of events.....	20
3.2.3: Sequence Diagram.....	20
Chapter 4. Dataset.....	22

4.1 RWF-2000 Dataset .....	22
4.2 Datasets Conclusion .....	23
Chapter 5. Implementation .....	25
5.1 Environment Setup & Tools.....	25
5.2 Packages and Libraries .....	25
5.3 Preprocessing Techniques .....	27
5.3.1 Resizing.....	27
5.3.2 Normalization.....	27
5.4 Models .....	28
5.4.1 CONVLSTM.....	28
5.4.2 C3D .....	31
5.4.3 3DCNN .....	35
5.4.4 Transformer-based .....	37
Chapter 6. Experiments & Results .....	40
6.1 ConvLSTM model.....	40
6.2 C3D model .....	43
6.3 3DCNN model.....	45
6.4 Transformer-based model.....	45
6.5 Results Summary.....	48
6.6 Comparative Analysis .....	49
Chapter 7. Data Analysis.....	50
7.1 Dimensionality Reduction.....	50
7.1.1 t-SNE.....	50
7.1.2 UMAP .....	51
7.2 Clustering .....	51
7.2.1 K-means .....	51
7.2.2 DBSCAN.....	52
Chapter 8. User Manual.....	56
8.1 Detailed Features – Portal User Flow.....	56

8.1.1 Login .....	56
8.1.2 Sign up .....	58
8.1.3 Delete account .....	60
8.2 Main Features .....	62
8.2.1 Upload video .....	62
8.2.3 SOS .....	65
8.3 Side Features .....	66
8.3.1 Violence Declaration .....	66
Chapter 9. Conclusion and Future Work .....	69
9.1 Conclusion .....	69
9.2 Future Work .....	70
References .....	71

# List of Figures

Figure 1. 1. Time Plan.....	2
Figure 2. 1. A schematic overview of the proposed method.....	8
Figure 2. 2. Structure of the two CNN-LSTM streams that comprise the proposed model. ....	8
Figure 2. 3. The structure of the Flow Gated Network .....	11
Figure 2. 4. DenseNet Architecture .....	14
Figure 2. 5. DenseNet architectures for ImageNet. ....	14
Figure 2. 6. shows the transformer encoder structure. ....	16
Figure 3. 1. System Architecture .....	18
Figure 3. 2. Use Case Diagram.....	19
Figure 3. 3. Sequence Diagram .....	21
Figure 4. 1. Video examples in the RWF-2000 dataset.....	22
Figure 4. 2. The pipeline of RWF-2000 dataset collection.....	23
Figure 5. 1. An example of resizing images. ....	27
Figure 5. 2. An example of normalizing images. ....	28
Figure 5. 3. Getting the input ready for the proposed model. ....	29
Figure 5. 4. Explains the stages of video processing before starting the training process .....	30
Figure 5. 5 The dense optical flow algorithm output can be encoded as the HSV color scheme.....	32
Figure 5. 6. shows the result of optical flow after converting to HSV .....	32
Figure 5. 7. An example of converting RGB image to gray image .....	33
Figure 5. 8. The structure of the Flow Gated Network .....	33
Figure 5. 9. The Architecture of the 3DCNN Model.....	35
Figure 5. 10. An example of center cropping of image .....	38
Figure 5. 11. Another example of center cropping of an image lost its importance .....	38
Figure 6. 1. Pre-Trained SepConLSTM-C model on RWF2000 train vs test accuracy .....	41
Figure 6. 2. Pre-Trained SepCon LSTM-C model on RWF2000 train loss vs test loss.....	42
Figure 6. 3. Confusion matrix of the test set of the SepConvLSTM-C model.....	42
Figure 6. 4. Trial 2 of C3D model accuracy vs. validation accuracy.....	43
Figure 6. 5. Trial 2 of C3D model loss vs. validation loss.....	44
Figure 6. 6. Confusion matrix of the test set of C3D model trial 1 .....	44
Figure 6. 7. Confusion matrix of a test set of 3DCNN model.....	45
Figure 6. 8. Confusion matrix of a test set of transformer model trial 1 .....	46
Figure 6. 9. Confusion matrix of a train set of transformer model trial 1.....	46
Figure 6. 10. Confusion matrix of a test set of transformer model trial 2 .....	47
Figure 6. 11. Confusion matrix of a train set of transformer model trial 2.....	47
Figure 7. 1. Representation of how K-means works.....	52
Figure 7. 2. Representation of how DBSCAN works .....	53
Figure 7. 3 t-SNE with K-means.....	53
Figure 7. 4. UMAP with DBSCAN .....	53
Figure 7. 5. K-means then applying t-SNE.....	54
Figure 7. 6. DBSCAN then applies t-SNE.....	55

## List of Tables

Table 2. 1. The detailed parameters of the model structure .....	13
Table 4. 1. A comparison between our proposed RWF-2000 dataset and previous datasets. ....	24
Table 5. 1. The detailed parameters of the model structure .....	36
Table 5. 2. The architecture layers of the transformer model. ....	39
Table 6. 1. Compare classification results for datasets.....	40
Table 6. 2. Reported accuracy on the RWF2000 Test subset.....	41
Table 6. 3. Compare classification results for best trials.....	43
Table 6. 4. Compare classification results for best trials.....	45
Table 6. 5. All results of all models for the RWF-2000 dataset .....	48
Table 6.6. Comparison between related work and our work on the test subset of RWF-2000 .....	49



## List of Abbreviations

Abbreviation	Stands for
BN	Batch Normalization
CNN	Convolutional Neural Network
C3D	Convolutional 3D
CPU	Central Processing Unit
ConvLSTM	Convolutional Long Short-Term Memory
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DL	Deep Learning
DenseNet	Densely Connected Convolution Neural Network
F1	F1 Score
GPU	Graphics Processing Unit
KNN	K-Nearest Neighbor
LSTM	Long Short-Term Memory
P	Precision
R	Recall
RGB	Red, Green, Blue
RNN	Recurrent Neural Network
SVM	Support Vector machine
t-SNE	t-Distributed Stochastic Neighbor Embedding
UI	User Interface
UMAP	Uniform Manifold Approximation and Projection

# **Chapter 1. Introduction**

In recent years, rates of violence have been increasing at an alarming rate -whether physical violence, domestic violence, gun violence, petty crimes, or any other form-, posing a grave threat to many individuals in society. Consequentially, many people have taken to the use of video surveillance cameras to remotely monitor their property. Despite being an accessible and efficient solution, it requires great human effort to keep track of all surveillance cameras 24/7 and does not guarantee immediate action to prevent any escalations.

## **1.1 Problem Definition**

While video footage provided by surveillance cameras is very useful for both crime minimization and prevention, it does not come at a cheap cost. The monitor on shift is required to stay attentive to several cameras at a time, for very long hours every day, which undermines the chance to make decisions in a short time. The monitor on shift is required to stay attentive to several cameras at a time, for very long hours every day, which undermines the chance to make decisions in a short time [1]. If, for any reason, the monitor's attention is divided, any violation committed would go undetected, and thus result in not only potential harm or even life losses but also grueling hours of manually winding back footage to find the culprit afterward.

## **1.2 Motivation**

Recent studies have shown that in 2021, the global homicide rate was estimated to be 5.8 per 100,000 people [2] and was not predicted to slow down. Surveillance cameras have been found to help reduce the occurrence of crimes by around 16% [3]. This rate could be further improved if those cameras detect violent events automatically and act without human intervention.

## **1.3 Objectives**

- Build a classification model that accurately detects violence in videos.
- Build a clustering model that can cluster the subtype of violence detected.
- Generate a brief report and send it to the person in charge.
- Develop a user-friendly, accessible application.

## 1.4 Methodology

Violence detection as a concept relies on a model's ability to detect human action in the first place, and then classify whether it is violent or not. While older research focused on various traditional machine-learning techniques, recent advancements have shown that deep-learning methods yield far better results [1].

We proposed a framework to solve the violence detection tasks that contain two modules: preprocessing of a video feed then processing the standardized video into a deep learning model. These DL models utilize a CNN, a class of artificial neural networks that has become dominant in various computer vision tasks, is used to identify and extract features from images through backpropagation, by building blocks, which consist of convolution layers, pooling layers, and fully connected layers. Another technique that was utilized was the ConvLSTM. In a traditional LSTM, an RNN, the model passes the previous hidden state to the next step of the sequence, therefore making it possible for old information to be retained to make future decisions. In order to utilize this advantage, as well as the CNN's abilities used for image processing, the ConvLSTM was designed to combine these two models to handle spatiotemporal data that involve both spatial and temporal dependencies. At last, we attempted to employ a transformer-based model. Transformers are a type of neural network architecture that transforms or changes an input sequence into an output sequence, through learning context and tracking relationships between sequence components.

## 1.5 Time Plan

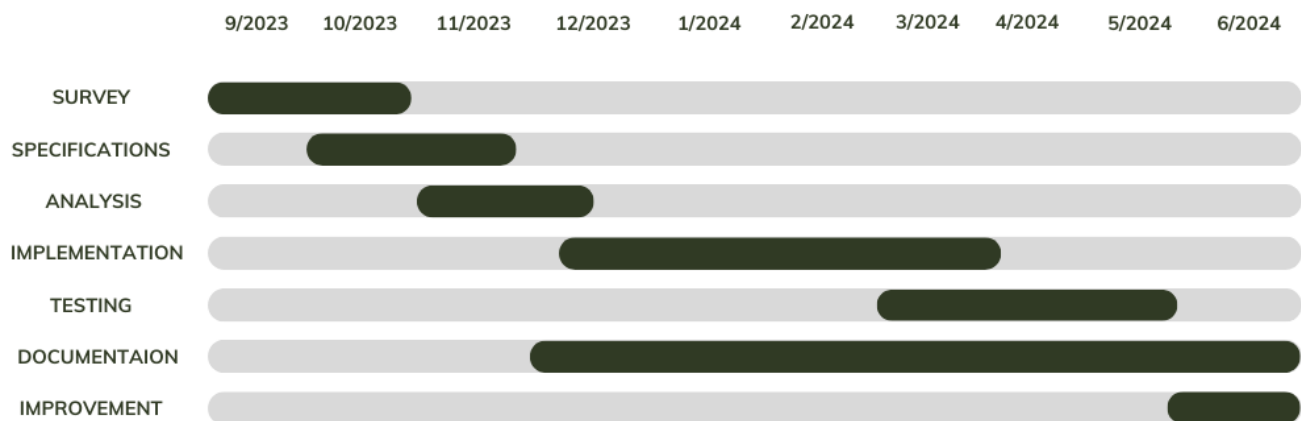


Figure 1.1 Time Plan

## 1.6 Thesis Outline

Chapter 2: Background

This chapter includes background about the project, basic concepts, and the related work according to our research.

### Chapter 3: Analysis & Design

This chapter includes an overview of the whole system along with the intended user, system architecture, analysis, and design.

### Chapter 4: Dataset

This chapter includes an overview of the dataset used in the system.

### Chapter 5: Implementation

This chapter describes the system functions with details about the implementation of the project's modules.

### Chapter 6: Results

This chapter includes some experiments we did through our work to improve the results.

### Chapter 7: Data Analysis

This chapter includes some attempts to classify violent videos based on the type of violence.

### Chapter 8: User Manual

This chapter talks about the needed packages and libraries that must be installed before using the application and shows the user how to use it.

### Chapter 9: Conclusion & Future Work

This chapter concludes our thesis and the results of our work. It also highlights potential directions of future work that may be done based on this project.

## Chapter 2. Background

### 2.1 Literature Review

Vidas Raudonis et al. [4] proposed a solution that aims to reduce reaction time in cases of harm to public health and safety by attracting the attention of human operators to the highlighted screen using computer vision and machine learning techniques. The model was trained for different datasets (RWF-2000, Hockey Fights, and Movie Fights), as the model is divided into three stages, which are The spatial feature extractor uses a U-Net-like network with MobileNet V2 network model as an encoder for static single frame feature extraction due to its high accuracy and smaller network size; to improve training effectiveness, The temporal feature extractor uses LSTM to capture sequential information between consecutive video slices, and The classifier, based on dense layers, annotates events as violent or nonviolent, and the model used the binary cross-entropy loss function for video classification. The experiments demonstrated good performance of the proposed model, which has 4,074,435 parameters and is computationally light and fast. Cross-validation was performed with five folds. The models achieved good results, with an average accuracy of  $82.0 \pm 3\%$  for the RWF-2000 dataset,  $96.1 \pm 1\%$  for the Hockey Fight dataset, and  $99.5 \pm 2\%$  for the Movie Fight dataset. However, this paper had some limitations, like the insufficient data frames from video clips, their challenges with increasing false alarm rates, and the high computational methods based on time complexity for real-time detection.

Juan C. San Miguel et al. [5] a novel deep-learning architecture for detecting violent crimes in surveillance videos. It used human pose extractors and change detectors as inputs, combining them using a novel method that used additions instead of multiplications. The architecture also accounted for spatial and temporal information using the ConvLSTM. The authors use different datasets (RWF-2000, Hockey Fight, Crowded, and Movie Fight) and propose a novel architecture for detecting violence in surveillance videos, aiming for computational efficiency, reduced parameters, and high model accuracy. They simplify the model's input by extracting essential information from human bodies and their interactions, combine features from interactions and temporal changes, and use ConvLSTM for efficient two-stream architectures, resulting in 5x fewer parameters and real-time accuracy. Final model accuracy evolution. After 50 epochs, the best model achieved 90.25% in the validation set and 92.37% in the training set.

Islam, Zahidul et al. [6] proposed a two-stream deep learning architecture for detecting aggressive human behaviors like fighting, robbery, and rioting, suitable for unmanned security monitoring systems and internet video filtration. The architecture uses Separable

Convolutional LSTM (which reduces computation and parameter count, producing localized Spatio-temporal features) and MobileNet (pre-trained on ImageNet dataset) as the CNN for extracting spatial features. The best model achieved 89.75% test accuracy for the RWF-2000 dataset.

Paolo Sernani et al. [1] presented a comparison of three different deep learning models on the AIRTLab dataset. (The dataset includes 350 full HD clips at 30 frames per second, which includes non-violent samples that can cause false positives.). The dataset is designed to test the performance against false positives, and the authors propose two transfer learning-based models and one "trained from scratch" model. The authors tested these models on the AIRTLab dataset to validate their performance and build a comparison with existing literature. By comparing the proposed models against the performance of end-to-end models based on pre-trained 2D CNNs, the authors provided an extended baseline of results for well-established networks on the AIRTLab dataset and the Hockey Fight and Crowd Violence datasets. The models achieved results with an average accuracy of  $97.15 \pm 1.5\%$  for the AIRTLab dataset,  $97.86 \pm 0.8\%$  for the Hockey Fight dataset, and  $99.06 \pm 0.5\%$  for the Crowd Violence dataset. The research presented promising deep learning-based models for violence detection but has limitations due to the lack of real violence in the proposed dataset. The models were validated on real videos from the Hockey Fight and Crowd Violence datasets. The model was built on previous work and violence detection research, but a systematic study on alternative hyperparameters and models is needed for more general results. should be evaluated before production, with sub-sequences of frames merged to maximize accuracy.

Manan Sharma et al. [7] that detecting actions in videos is a complex task that requires reasoning, and deep learning algorithms and cameras can replace human reasoning power, providing an end-to-end model for video classification. They used a combination of CNN and LSTMs for video classification. The ConvLSTM cell, a type of LSTM cell, performs convolution operations inside the LSTM cell, capturing underlying spatial features in multi-dimensional data. This model combines pattern recognition of ConvNets with the memory properties of pure LSTM networks, aiming to find patterns in image sequences. Three different datasets were used: KTH, Violent-Flows, and Hockey. KTH achieved 100% accuracy, Violent Flows achieved 80%, and Hockey achieved 87.5% accuracy, with four reducing points in the learning curve. The CNN model that gives good results among all three is: ResNet50 CNN has an accuracy of 89.9%, InceptionV3 CNN has an accuracy of 88.6%, and VGG19 CNN has an accuracy of 79.3%.

## 2.2 Techniques

### 2.2.1 ConvLSTM

One type of deep learning algorithm created especially for image processing and recognition applications is CNN. From raw input photos, CNNs may automatically learn representations of hierarchical features less preprocessing is needed than with other classification models. CNNs may learn even more complex features by stacking many layers of convolution and pooling. This leads to excellent accuracy in tasks like segmentation, object identification, and picture classification.

#### 2.2.1.1 Separable Convolutional LSTM

The proposed method aims to complete the creation of a comprehensive trainable neural network that can efficiently capture long-term features of space and time to identify violence as in [6]. It works in a computationally efficient manner. To do this, work has been completed on a new and highly effective network to detect two-way violence. An easy method to suppress non-moving background information that propagates the capture of discriminative characteristics and emphasizes body movements in frames has been devised. This section begins with a description of the model's essential component, the separable convolutional LSTM. The input preparation stages that are incorporated into our pipeline have already been covered. Lastly, a summary of the suggested network architecture and the many fusion strategies that have been created and are now in use is given.

- Separable Convolutional LSTM

The Separable Convolutional LSTM (SepConvLSTM) combines the efficiency of depthwise separable convolutions with the temporal modeling capability of ConvLSTM. Let's break down the key aspects and equations of the SepConvLSTM cell as described in the provided text.

- Depthwise Separable Convolutions

- **Depthwise Convolution:** Each input channel is convolved separately with a different filter.
- **Pointwise Convolution (1×1 Convolution):** A 1×1 convolution is applied to combine the depthwise convolution outputs across channels.
- **Efficiency:** This approach reduces the computational cost by a factor of

$$\frac{1}{N} + \frac{1}{K^2} \quad (2.1)$$

where N is the number of output channels and K is the kernel size.

- SepConvLSTM Cell Operations

The equations (4) describe the operations inside a SepConvLSTM cell, where convolution operations are replaced with depthwise separable convolutions. Here's the detailed explanation

- Input Gate ( $i_t$  )

$$i_t = \sigma(1 * 1W_i^x * (W_i^x \otimes x_t) + 1 * 1W_i^h * (W_i^h \otimes h_{t-1}) + b_i)$$

- Forget Gate ( $f_t$  )

$$f_t = \sigma(1 * 1W_f^x * (W_f^x \otimes x_t) + 1 * 1W_f^h * (W_f^h \otimes h_{t-1}) + b_f)$$

- Cell State Update ( $\bar{c}_t$  )

$$\bar{c}_t = \tau(1 * 1W_c^x * (W_c^x \otimes x_t) + 1 * 1W_c^h * (W_c^h \otimes h_{t-1}) + b_c)$$

- Output Gate ( $o_t$  )

$$o_t = \sigma(1 * 1W_o^x * (W_o^x \otimes x_t) + 1 * 1W_o^h * (W_o^h \otimes h_{t-1}) + b_o)$$

- Final Cell State ( $C_t$  )

$$c_t = f_t \otimes c_t - 1 + i_t \otimes c_t$$

- Hidden State ( $h_t$  )

$$h_t = o_t \otimes (c_t) \tag{2.2}$$

Here,  $*$  represents convolution,  $\otimes$  represents the Hadamard product,  $\sigma$  represents sigmoid activation,  $\tau$  represents tanh activation and represents  $\otimes$  depthwise convolution.  $1 \times 1W$  and  $W$  are pointwise and depthwise kernels respectively. Memory cell  $C_t$ , hidden state  $h_t$  and the gate activations  $f_t$ ,  $i_t$  and  $o_t$  are all 3-dimensional tensors.

The SepConvLSTM cell combines the efficiency of depthwise separable convolutions with the spatiotemporal modeling capabilities of ConvLSTM, making it a compact and lightweight choice for encoding spatiotemporal feature maps in various applications such as video analysis and time-series forecasting.



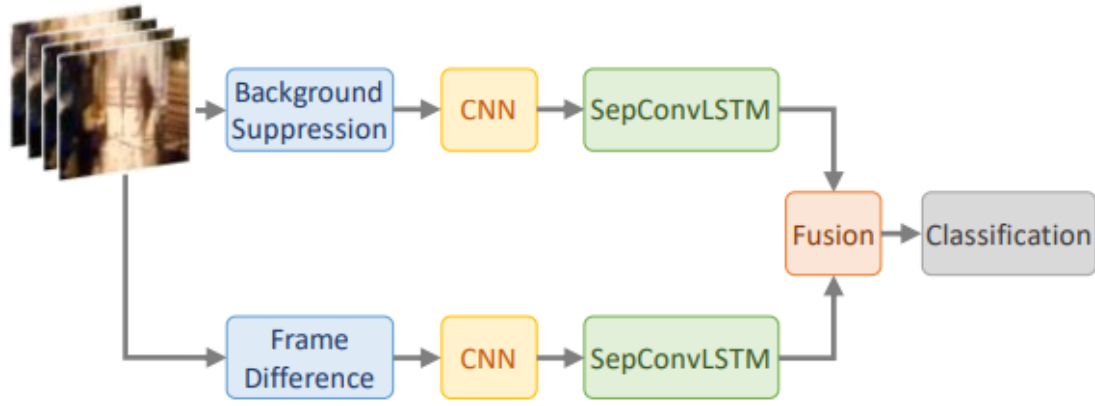


Figure 2. 1. A schematic overview of the proposed method.

SepConvLSTM is constructed by replacing convolution operation at each gate of ConvLSTM with a depthwise separable convolution that enables producing robust long-range Spatiotemporal features while using fewer parameters. **Figure 2.1**, the proposed pipeline contains two streams consisting of CNN and SepConvLSTM modules. Background Suppression and Frame Difference are preprocessing modules. The outputs of the two streams are combined to produce robust spatiotemporal features.

- **Separable Convolutional LSTM Architecture**

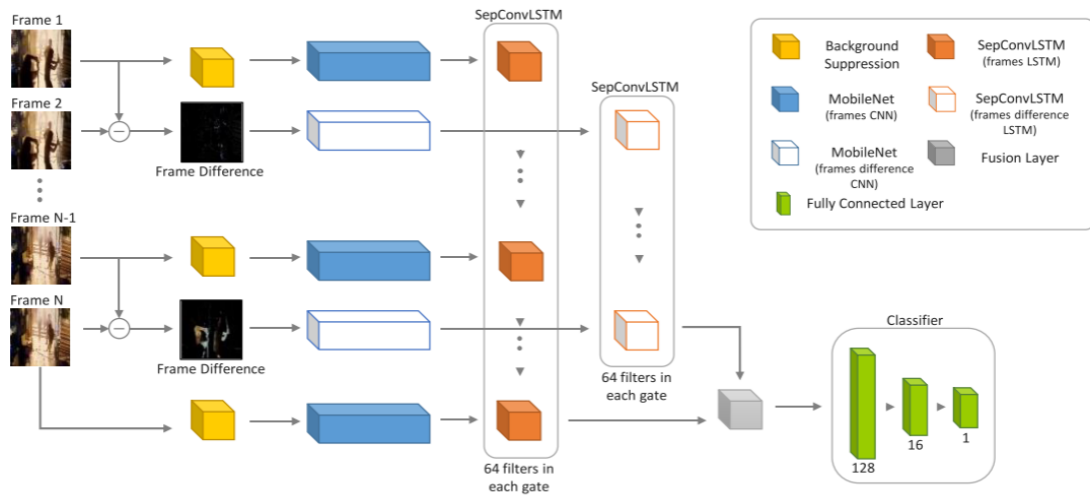


Figure 2. 2. Structure of the two CNN-LSTM streams that comprise the proposed model.

Each stream is composed of a condensed version of the MobileNet module that generates spatial attributes from each input time step. These properties are sent to the SepConvLSTM cell of each stream so that it may build spatiotemporal encodings. The outputs from each stream are sent into a fusion layer, which combines them before sending them to the classifier network.

As shown in Figure 2.2, the suggested network is composed of two separate streams with similar designs. At every video time step, a 2D convolutional network in each stream collects spatial data. After an LSTM layer encodes these spatial attributes, it generates spatiotemporal feature maps, which are subsequently sent to the classification layers. The model receives video frames with the backdrop suppressed one after the other on the first stream. Following the processing of each frame, the spatio-temporal features were extracted from the last time step of the hidden state of LSTM. The second stream is processed in the same way, except here the inputs are the differences between neighboring frames. By eliminating the computing burden of calculating optical flow, frame differences provide an effective approximation of optical flow. While the other stream primarily concentrates on information based on spatial appearance, the frame difference stream learns to encode temporal changes, capturing the motion in between frames. When the output characteristics from the two streams are integrated, strong spatiotemporal feature maps that can differentiate between violent and non-violent videos are produced. Using the pre-trained ImageNet dataset [8], MobileNetV2( $\alpha = 0.35$ ) [9] was employed as a CNN to extract spatial features, with  $\alpha$  being the width multiplier. The last 30 layers of MobileNet models turned out to be redundant in our early tests, so they were shortened. Pretraining expedites training and enhances generalization. From CNN's output feature maps, Local spatiotemporal features were generated separable convolutional LSTM (SepConvLSTM). While SepConvLSTM has not been investigated for action classification tasks, it has been employed in the past to speed up video segmentation tasks. Shape  $224 \times 224 \times 3$  frames are fed into the model. CNN collects spatial characteristics of shape  $7 \times 7 \times 56$  from each stream. Since SepConvLSTMs with 64 filters were used, each produced a feature map with dimensions of  $7 \times 7 \times 64$ . Following a window size (2,2) Max-Pooling layer, a Fusion layer explained in the next section is used to fuse the output feature maps from the two streams. Subsequently, fully linked layers get the aggregated feature maps for classification. The activation of LeakyRelu is applied between completely linked layers. Finally, the outputs of the final layer are used to compute the binary cross-entropy loss. To examine each stream's contribution, Single-stream versions of our model have also been tried. The fusion layer and other stream layers are simply deleted to create single-stream variants of the proposed model.

In the end, three fusion algorithms were developed to combine the two routes' output feature maps. These three approaches make up the SepConvLSTM-M, SepConvLSTM-C, and SepConvLSTM-A versions of suggested model [6]. The following describes the fusion layers for these three variations.

- SepConvLSTM-M

A LeakyRelu activation layer is applied to the frame streams' output. In contrast, a Sigmoid activation layer processes the feature maps from the frame difference stream. After that, Each element is multiplied independently to obtain the final feature maps.

$$F_{fused} = LeakyRelu(F_{frames}) \otimes Sigmoid(F_{diff}) \quad (2.3)$$

Here,  $F_{frames}$  and  $F_{diff}$  denotes the feature maps from frames stream and frame difference stream respectively.  $F_{fused}$  is the output feature map of the Fusion layer.

- SepConvLSTM-C

To send the data to the classification layers, we just concatenate the two output features of the two streams.

$$F_{fused} = Concat(F_{frames}, F_{diff}) \quad (2.4)$$

Here, the Concat function concatenates  $F_{frames}$  and  $F_{diff}$  along the channel axis.

- SepConvLSTM-A

The output feature maps of the two streams are added element by element.

$$F_{fused} = (F_{frames}) \oplus (F_{diff}) \quad (2.5)$$

Here,  $\oplus$  refers to element-wise addition operation combining the output feature maps of the two streams.

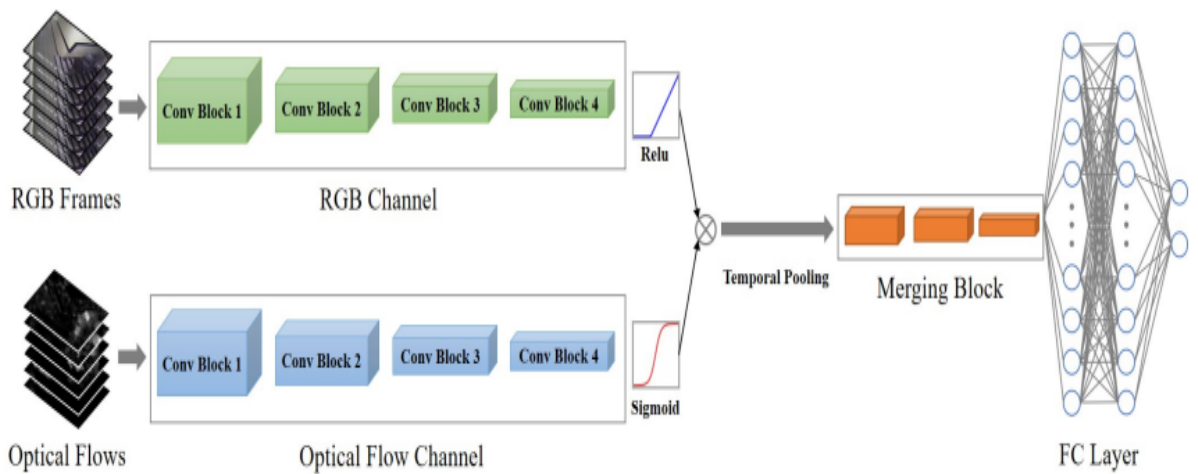
### 2.2.2 C3D

C3D is an architecture designed to process and analyze 3D data, such as video frames or volumetric medical images, making it an invaluable tool for tasks such as action recognition, video analysis, and medical image classification.

One of the key advantages of C3D is its ability to preserve the temporal dynamics of data, allowing it to effectively process video sequences and capture the evolution of features over time. This makes it a powerful tool for applications such as video action recognition, where understanding how objects and actions change over time is crucial [10].

C3D and 3DCNN are both types of convolutional neural networks designed to process spatio-temporal data (data that includes both spatial and temporal components, which means that the data has information about the location (space) and the time (temporal) when the event or observation occurred), such as videos. While C3D is specialized for action recognition tasks, 3DCNNs can be applied to a wide range of spatiotemporal data analysis tasks.

The aim is to design a temporal pooling mechanism achieved by network self-learning. The structure of our proposed model with four parts: the RGB channel, the Optical Flow channel, the Merging Block, and the Fully Connected layer, as shown in Figure 2.3.



**Figure 2. 3. The structure of the Flow Gated Network**

RGB channel and Optical Flow channel consist of cascaded 3DCNNs, and they have consistent structures so that their output can be fused. Merging Block is also composed of basic 3DCNNs, which process information after self-learned temporal pooling. Finally, the Fully connected layers generate output.

The highlight of this model is that it utilizes a branch of the optical flow channel to help build a pooling mechanism. Relu activation is adopted at the end of the RGB channel, while the sigmoid function is placed at the end of the optical flow channel. Then, outputs from RGB and optical flow channels are multiplied together and processed by temporal max

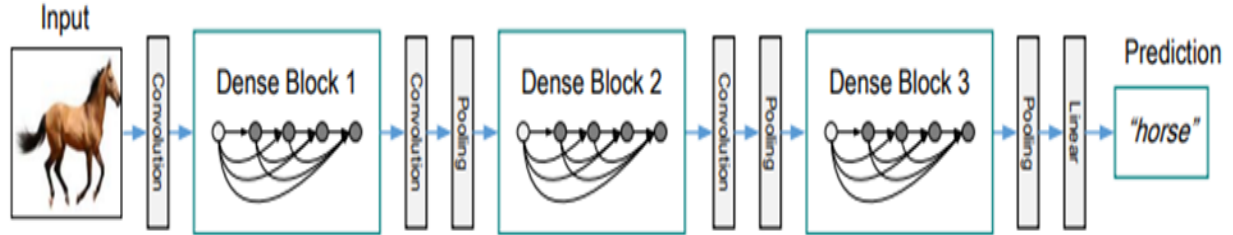
pooling. Since the output of the sigmoid function is between 0 and 1, it is a scaling factor to adjust the output of the RGB channel. Meanwhile, as max pooling only reserves local maximum, the outcome of being an RGB channel multiplied by one will have a larger probability of being retained, and the value multiplied by zero is more likely to be dropped. This mechanism is a kind of self-learned pooling strategy that utilizes a branch of optical flow as a gate to determine what information the model should preserve or drop.

**Table 2. 1. The detailed parameters of the model structure**

Block name	Type	Filter Shape	Number Of Repeats
RGB/Flow channels	Conv3d Conv3d MaxPool3d	1×3×3@16 3×1×1@16 1×2×2	2
	Conv3d Conv3d MaxPool3d	1×3×3@32 3×1×1@32 1×2×2	2
Fusion and pooling	Multiply	None	1
	MaxPool3d	8×1×1	1
Merging block	Conv3d Conv3d MaxPool3d	1×3×3@64 3×1×1@64 2×2×2	2
	Conv3d Conv3d MaxPool3d	1×3×3@128 3×1×1@128 2×2×2	1
Fully-connected layers	FC layer	128	1
	FC layer	32	1
	SoftMax	2	1

## 2.2.4 DenseNet-121

A DenseNet is a type of convolutional neural network that utilizes dense connections between layers, through Dense Blocks, where all layers are connected (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its feature maps to all subsequent layers.



**Figure 2. 4. DenseNet Architecture**

Since each layer receives feature maps from all preceding layers, the network can be thinner and more compact, i.e., the number of channels can be fewer. The growth rate  $k$  is the additional number of channels for each layer. So, it has higher computational efficiency and memory efficiency.

In Figure 2.4,  $1 \times 1$  Conv followed by  $2 \times 2$  average pooling is used as the transition layer between two contiguous dense blocks. Feature map sizes are the same within the dense block so that they can be concatenated together easily. At the end of the last dense block, global average pooling is performed, and then a SoftMax classifier is attached.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	$112 \times 112$	$7 \times 7$ conv, stride 2			
Pooling	$56 \times 56$	$3 \times 3$ max pool, stride 2			
Dense Block (1)	$56 \times 56$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	$56 \times 56$	$1 \times 1$ conv			
	$28 \times 28$	$2 \times 2$ average pool, stride 2			
Dense Block (2)	$28 \times 28$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	$28 \times 28$	$1 \times 1$ conv			
	$14 \times 14$	$2 \times 2$ average pool, stride 2			
Dense Block (3)	$14 \times 14$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	$14 \times 14$	$1 \times 1$ conv			
	$7 \times 7$	$2 \times 2$ average pool, stride 2			
Dense Block (4)	$7 \times 7$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	$1 \times 1$	$7 \times 7$ global average pool			
		1000D fully-connected, softmax			

**Figure 2. 5. DenseNet architectures for ImageNet.**

DenseNet architecture maximizes information flow between layers by connecting all layers directly with each other. Each layer receives additional input from preceding layers and passes on its feature maps to subsequent layers. This approach differs from ResNets in that it does not combine features through summation but instead concatenates them. This results in  $L(L+1) / 2$  connections in an L-layer network, requiring fewer parameters than traditional convolutional networks. The benefits of using DenseNet include strong gradient flow, parameter & computational efficiency, more diversified features, and low complexity features.

## **2.2.5 Transformer**

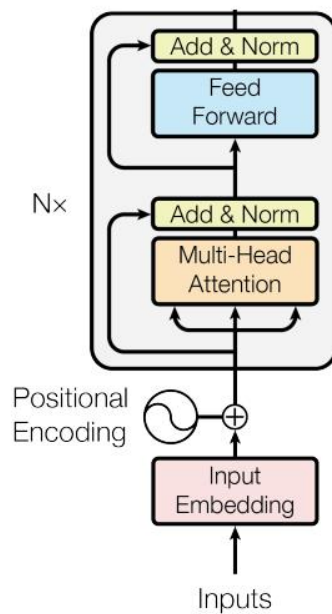
### **2.2.5.1 Positional Encoding**

By embedding positional information, it allows the model to understand the order of elements within a sequence. This layer is particularly important in transformer models, where it compensates for the lack of inherent order in self-attention mechanisms. Its architecture involves initializing an embedding layer, building it according to the input shape, and adding positional embeddings to the input data during the forward pass. This process enhances the model's ability to capture and utilize positional relationships in the data, leading to better performance on tasks involving sequences, as shown in Figure 2.6.

### **2.2.5.2 Encoder**

Transformer encoders are used in various tasks involving sequential data, such as NLP and time series analysis. The key components include multi-head self-attention, a feed-forward neural network, and layer normalization, as shown in Figure 2.6.





**Figure 2. 6. shows the transformer encoder structure.**

**The multi-head-attention mechanism** allows the model to focus on different parts of the input sequence, enabling it to capture complex dependencies.

**Feed-forward networks** (the dense projection layers) add non-linearity and allow for richer representations.

**Layer normalization** helps stabilize and speed up the training process by normalizing inputs across the features.

This architecture in Figure 2.6 allows the model to capture complex dependencies and relationships within the input sequence, making it highly effective for tasks involving sequential data. The use of residual connections and layer normalization further enhances training stability and model performance.

## **Chapter 3. System Architecture**

### **3.1 System Overview**

Violence Detection app is a mobile application designed to help detect violence spread through videos and live streams using advanced machine learning models. The system consists of several main components:

- UI (application)
- Backend Server
- DL Model

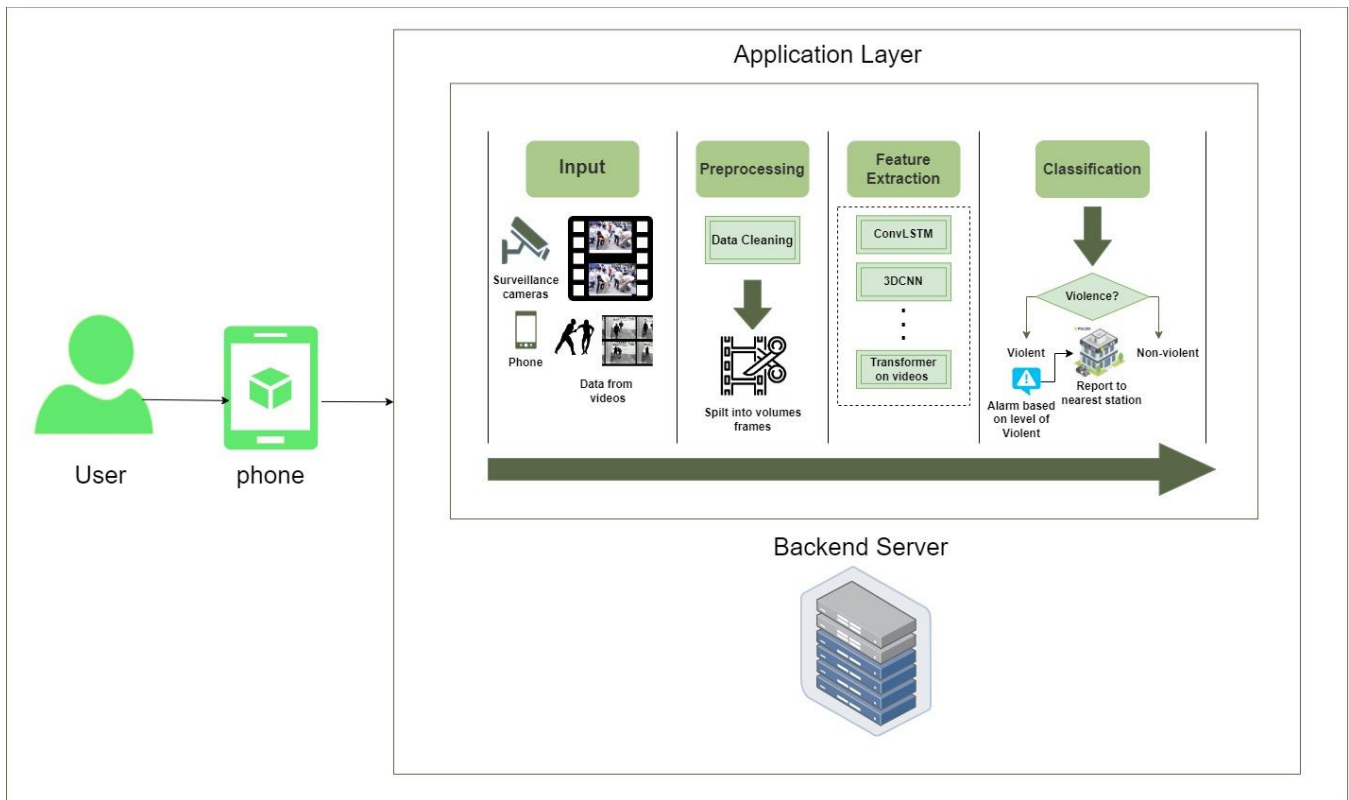
The user interface is the application with which the user interacts in terms of registering and using the application's features. The back and server are responsible for processing the data and sending it from the application to the model, which in turn performs the process of analyzing the data to discover whether there was violence or not.

The application features three main components:

- Detect violence by uploading a video
- Exposing violence through live broadcasting
- Reveal the type of violence in the video

#### **3.1.1 System Architecture**

The structure of Violence Detection applications for detecting violence. The main component is detecting violence through the mobile phone camera by recording frames through the application and sending them to the model via the back-end server to complete the detection process. Then the server sends the output to the application to be displayed to the user, avoiding storing the data in the Database. So that the discovery process takes place as quickly as possible.



**Figure 3. 1. System Architecture**

### 3.1.2 System Users

#### A. Intended Users:

Violence Detection mobile applications are designed for people and institutions that want to detect violence without human intervention, such as government institutions, schools, and parents with their children.

The application is designed to provide a set of features and resources to help distinguish between different types of violence and determine whether there is violence or not.

#### B. User properties:

The end user must have knowledge of how to use a mobile phone, record data properly, and upload and capture videos.

## 3.2 System Analysis & Design

### 3.2.1 Use Case Diagram



**Figure 3. 2. Use Case Diagram**

#### Description of Use Cases

##### Actors:

**User:** Interacts with the Violence Detection application to record, upload video clips, start living broadcasting, and use the rest of the application's features.

**Violence Detection application:** An interface through which the user uploads videos, starts live broadcasts, and receives notifications.

**Backend Server:** Handles recording, video processing, violence detection, and reporting.

##### Use cases:

**Register Account:** The user sends the data required to create an account in the application. This includes the user and the Violence Detection app.

Login: The user logs in to the Violence Detection application. This includes the user and the backend server.

Reset Password: The user changes his password. This includes the user and the backend server.

Video Upload: The user uploads a pre-recorded video for analysis. This includes the User App and Violence Detection.

Start Live: The user starts a live stream to reveal violence in real time. This includes the User App and Violence Detection.

Process Video: Violence Detection processes the video before sending it for analysis. This includes the Violence Detection application and backend server.

Classification for violence: The backend server classification the processed video to detect violence. This includes the backend server only.

User notification: The backend server notifies the user if violence is detected. This includes the backend server and the user.

Generate Report: The backend server generates a detailed report on the detected acts of violence. This includes the backend server and the user.

### **3.2.2: Flow of events**

- The user uploads a video clip or starts a live broadcast.
- Video pre-processing.
- The application inserts the processed video into the model.
- The model processes the input video and outputs the classification.
- The classification output is sent back to the application.
- The application displays the output to the user.
- If the classification is violent, the application warns the user and sends a report.

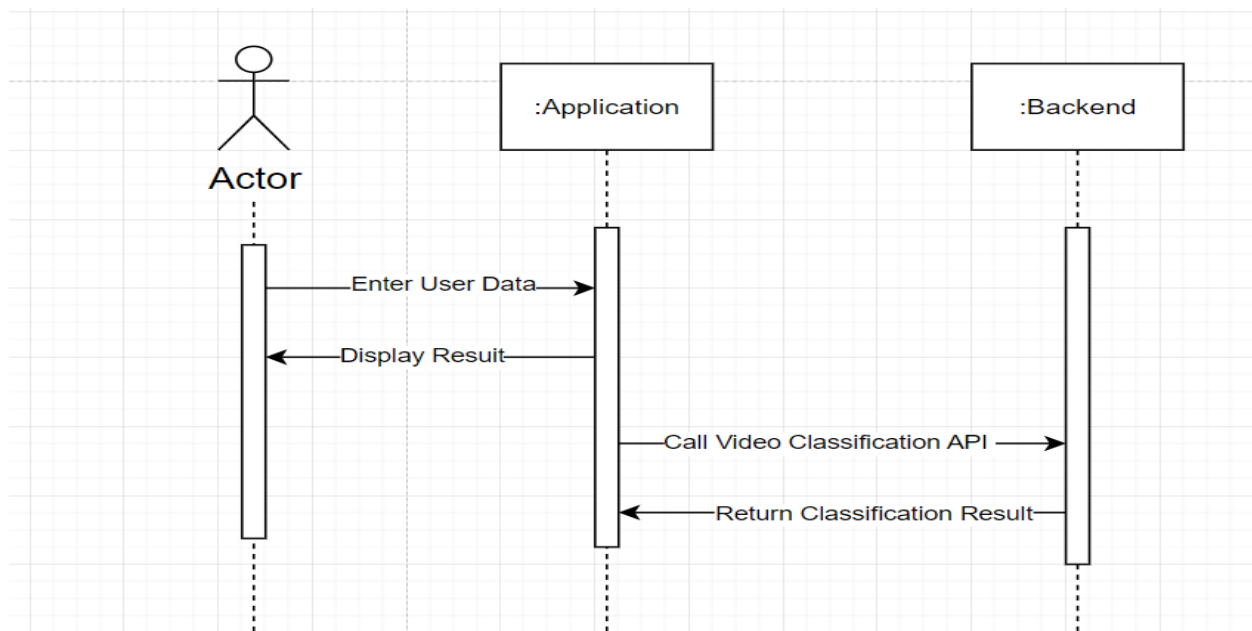
### **3.2.3: Sequence Diagram**

User: Sends data to the application.

Violence Detection Application: Displays a login success or failure message.

Backend Server: Analyzes the processed video frames of violence using a machine learning model.

Backend Server: Detects violence or no violence.



**Figure 3. 3. Sequence Diagram**

## Chapter 4. Dataset

The definition of video-based violence detection is detecting violent behaviors in video data; the availability of a suitable dataset plays a vital role in training and evaluating a violence detection system.

There are two kinds of video datasets for violence detection: trimmed and untrimmed. The videos in the trimmed datasets are all short clips with a length of several seconds, and each one of them has a video-level annotation. While the videos in untrimmed datasets usually have a longer duration, Furthermore, the start time and end time of violent activities have frame-level annotations.

### 4.1 RWF-2000 Dataset

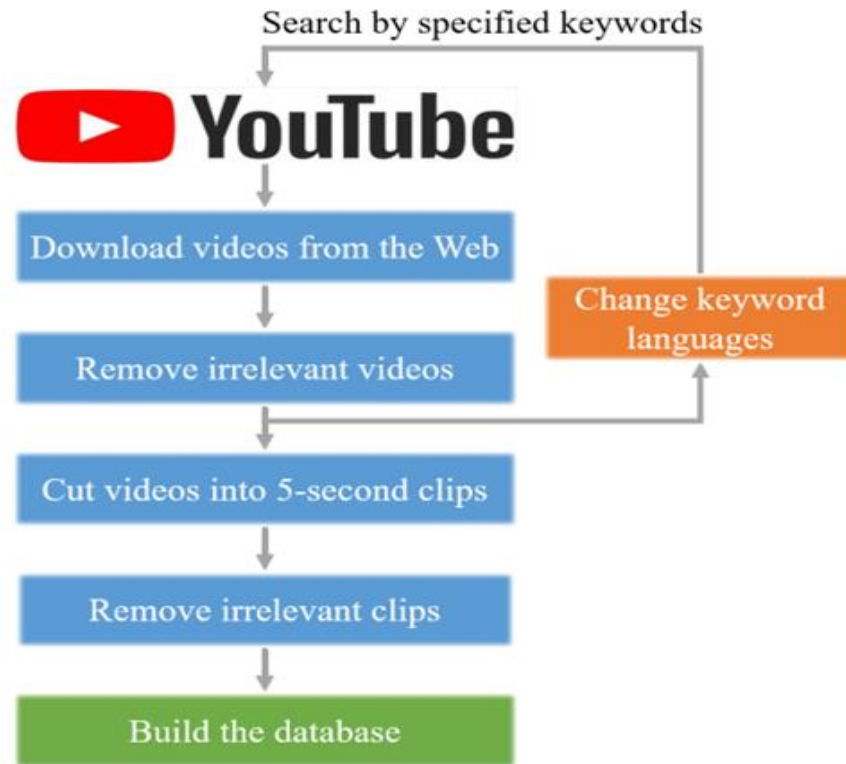
The RWF (Real-World Fight) [10] dataset was utilized for the development and evaluation of this violence detection system. The RWF dataset is a widely recognized benchmark dataset specifically designed for violence detection tasks in real-world scenarios (Figure 4.5.1) demonstrates some video examples in the RWF-2000 dataset.



**Figure 4. 1. video examples in the RWF-2000 dataset.**

It is collected from the YouTube platform, which consists of 2,000 video clips captured by surveillance cameras in real-world scenes (Figure 4.5.2) illustrates the pipeline of data collection. Firstly, the YouTube website is searched using a set of keywords related to violence, such as "real fights," "violence under surveillance," and "violent events." A list of URLs is obtained as a result. Subsequently, a program is employed to automatically

download videos from the obtained links. Each downloaded video is then checked to remove any irrelevant ones. To enhance the diversity of the dataset, the above procedures are repeated by changing keywords in various languages, including English, Chinese, German, French, Japanese, Russian, and more. The selection criteria do not impose limitations on the types of violence captured, encompassing any form of subjectively identified violent activity, such as fighting, robbery, explosion, shooting, blood, and assault. Following the data collection process, numerous raw videos are obtained. Each video is subsequently segmented into 5-second clips with a frame rate of 30 FPS. Finally, meticulous scrutiny is applied to delete noisy clips that contain unrealistic and non-monitoring scenes. Additionally, each remaining clip is annotated as either "violent" or "non-violent."



**Figure 4. 2. The pipeline of RWF-2000 dataset collection.**

## 4.2 Datasets Conclusion

The hockey fights dataset consists of 1000 videos (500 violent and 500 nonviolent) collected from various footage of ice hockey matches. Each video in the dataset contains 50 frames, capturing different moments during the matches. The movie's fights dataset has 200 video clips (100 violent and 100 nonviolent) focusing on scenes depicting fights in movies. The crowd fights dataset consists of 346 MP4 video files with an average length of 5.63 s. All the clips feature a 1920×1080-pixel resolution and a 30-fps frame rate. The



UCF-crime dataset consists of 1900 long and untrimmed real-world surveillance videos, with 13 realistic including Abuse, Arrest, Arson, Assault, Road Accident, Burglary, Explosion, Fighting, Robbery, Shooting, Stealing, Shoplifting, and Vandalism. These datasets are mainly composed of videos captured in a single scene, performed by actors, or extracted from edited movies. Only a few parts of them are from real events, but the RWF-2000 dataset is composed of videos captured from real-life situations, also these datasets have achieved state-of-the-art performance in violence detection, the RWF-2000 dataset did not achieve very high accuracy, so this is the main dataset that we used.

**Table 4. 1. A comparison between our proposed RWF-2000 dataset and previous datasets.**

<b>Dataset</b>	<b>Data scale</b>	<b>Length(sec)</b>	<b>Resolution</b>	<b>Scenario</b>
<b>Hockey fights[11]</b>	<b>1000 Clips</b>	<b>1.6 - 1.96</b>	<b>360 x 288</b>	<b>Hockey game</b>
<b>Movies fights[11]</b>	<b>200 Clips</b>	<b>1.6 - 2</b>	<b>720 x 480</b>	<b>Movie</b>
<b>Crowd fights[12]</b>	<b>246 Clips</b>	<b>1.04 - 6.52</b>	<b>1920×1080</b>	<b>Natural</b>
<b>UCF-Crime[13]</b>	<b>1900 Clips</b>	<b>60 - 600</b>	<b>Variable</b>	<b>Surveillance</b>
<b>RWF-2000</b>	<b>2000 Clips</b>	<b>5</b>	<b>Variable</b>	<b>Surveillance</b>

# Chapter 5. Implementation

## 5.1 Environment Setup & Tools

Several scripts were compiled together to run our model, all of which are written in Python, because it is the easiest, most helpful language to use due to its libraries, some of which are explicitly designed to help develop deep learning and machine learning models. We used “Google Colab” notebooks for our scripts due to the high processing needs of some of these scripts.

The application was created using Dart and Flutter, considering they were the easiest and fastest to learn, as well as the most compatible approach to connecting the model to the application. As a result, we were successful in creating a running demo quickly. Lastly, Android Studio was used to implement and debug our application.

### Google Colab specs:

- Intel Xeon CPU with 2 vCPUs
- NVIDIA Tesla T4 GPU with 16GB of VRAM
- NVIDIA L4 Tensor Core GPU with 24GB of VRAM
- DISK 78.2GB

### PyCharm on a Local PC:

- 4 GB NVIDIA RTX3090 TI
- INTEL CORE i7 10TH GEN
- 16GB of RAM

## 5.2 Packages and Libraries

### TensorFlow

TensorFlow is an end-to-end open-source platform for machine learning (ML). It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

### OpenCV

OpenCV is a library of programming functions aimed at real-time computer vision. It has 2500 algorithms, extensive documentation, and sample code for real-time computer vision.

### Keras

Keras offers consistent & simple APIs, minimizes the number of user actions required for

common use cases, and provides clear & actionable error messages. It also has extensive documentation and developer guides.

## **OS**

The OS module in Python's standard library provides functions for interacting with the operating system. This module provides a portable way of using operating system-dependent functionality.

## **Numpy**

NumPy brings the computational power of languages like C and Fortran to Python, a language much easier to learn and use.

## **Pandas**

Pandas allow us to analyze big data and make conclusions based on statistical theories, clean, messy data sets, and make them readable and relevant, it has functions for analyzing, cleaning, exploring, and manipulating data.

## **Typing**

The typing module provides runtime support for typing hints in Python. It allows you to specify the expected data types of variables, function parameters, and return values.

## **ColabCode**

A library that makes it easy to run a development server, like Fast API, directly within Google Colab notebooks by using ngrok.

## **Shutil**

The shutil module provides high-level file operations like copying and removing files.

## **Pyngrok**

The pyngrok library provides a Python wrapper for ngrok, which creates a secure tunnel to localhost, making your local server accessible over the internet.

## **Sklearn**

Scikit-Learn is also known as Sklearn. It's free and the most useful machine-learning library for Python. Sklearn library comes loaded with a lot of features, such as classification, regression, clustering and dimensionality reduction algorithms, including k-Means, KNN, support vector machines (SVM), and decision trees. It also supports Python numerical and scientific libraries NumPy and SciPy

## Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things and hard things possible.

## Umap

Umap (Uniform Manifold Approximation and Projection) is a dimensionality reduction technique that can be used for visualization similar to t-SNE but is faster and scales better to large datasets.

## Uvicorn

Uvicorn is a lightning-fast ASGI server implementation, using uvloop and httptools.

## 5.3 Preprocessing Techniques

### 5.3.1 Resizing

Videos are all different sizes, and this is a problem for deep learning. We fed the model videos as a volume or frame by frame; they all need to be the same size. So, we need to add a transform that will resize these images to the same size as shown in Figure 5.1. If the input images are very large, shrinking the size of these images will decrease the amount of time needed to train the model without significantly affecting model performance.



Figure 5. 1. An example of resizing images.

### 5.3.2 Normalization

Normalization is a fundamental preprocessing step that enhances the quality and consistency of video data and is used to transform features into a similar scale to speed up model learning, as shown in Figure 5.2.

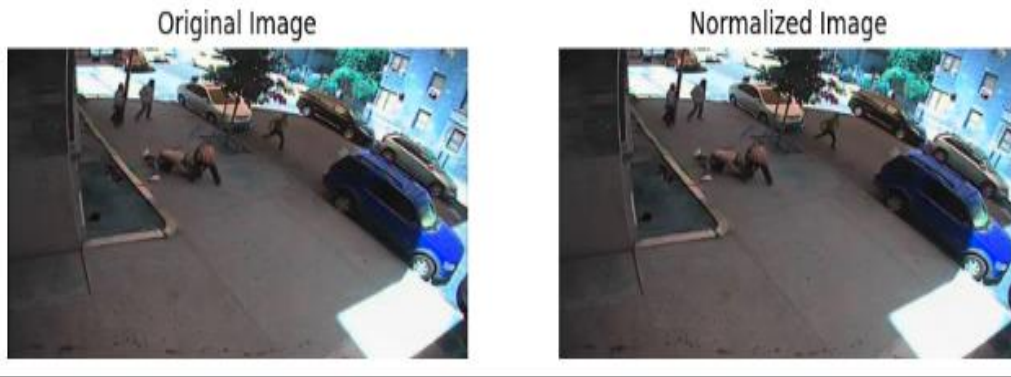


Figure 5. 2. An example of normalizing of images.

## 5.4 Models

### 5.4.1 CONVLSTM

#### 5.4.1.1 Additional Preprocessing

- Uniform Sampling

Uniform sampling can be used to reduce the number of frames for subsequent processing tasks such as training machine learning models, video summarization, or other analysis purposes. This involves selecting a subset of frames from the original video at regular intervals, ensuring an even distribution of frames across the entire video.

- Padding

Padding refers to the process of adding extra pixels around the edges of video frames. Padding is often used to ensure that all frames have the same dimensions, which is crucial for many video processing tasks, especially those involving machine learning models that require fixed-size inputs. Padding can also be used to prevent boundary artifacts in operations like convolution.

- Cropping Center

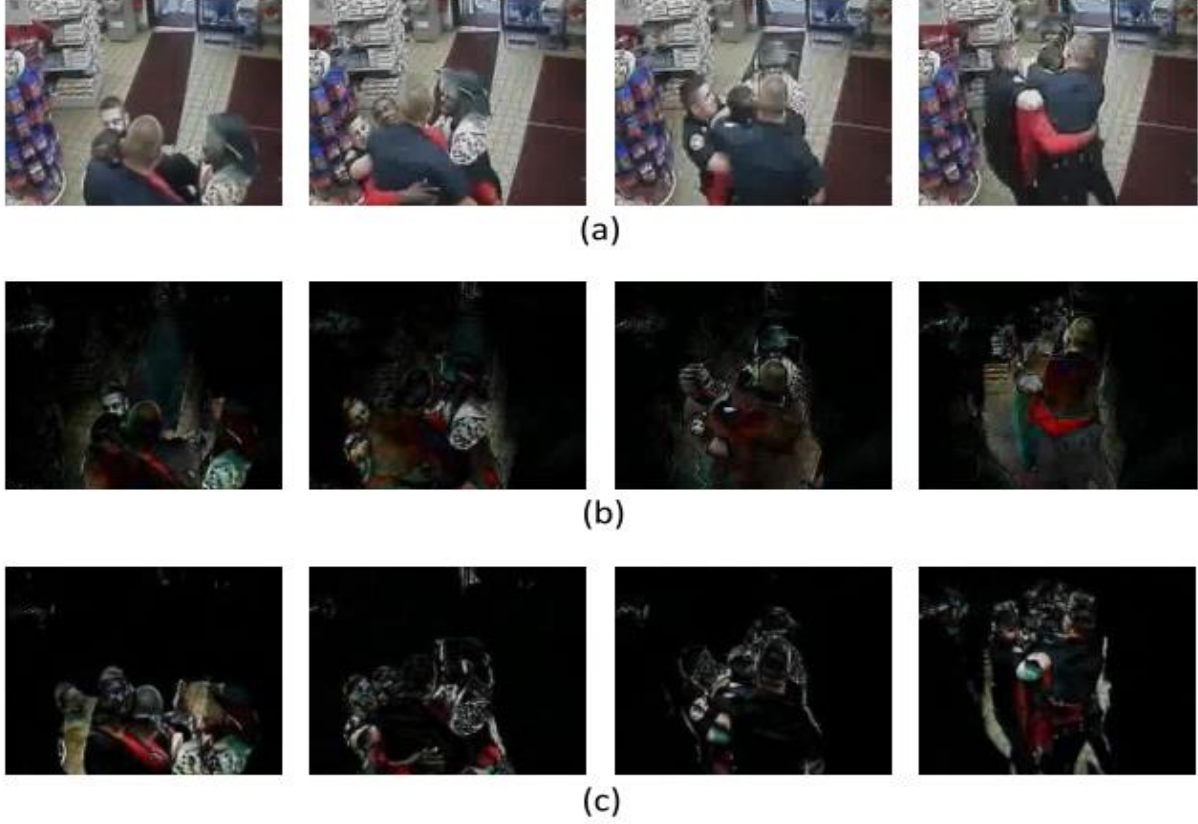
Cropping the center of video frames is used to focus on the most relevant part of the image, which often contains the main subject or action. This can reduce the computational load and remove unnecessary background information, making the analysis more efficient and potentially improving the performance of machine learning models.

- Background suppression and Frame differences

We feed one stream of our network with the difference between two neighboring frames as an input, which pushes the model to encode temporal changes between the neighboring frames, enhancing the capture of motion information. Their efficacy has been shown in

previous research [6], [14], and [15]. Frame disparities are a good stand-in for computationally expensive optical flow.

$$fd_i = frame_{i+1} - frame_i \quad (5.1)$$



**Figure 5. 3. Getting the input ready for the proposed model.**

In (a), keyframes from an example video clip are shown. (b) illustrates the effect of applying background suppression to the (a) video frames. The final row (c) shows the time steps of the frame difference extracted from the video clip of (a).

In equation 5.1, it  $frame_i$  shows the  $i$ th frame, which  $fd_i$  represents the  $i$ th time-step of the frame difference. A  $k$ -frame video clip results in a  $k-1$  time-step frame difference accordingly. Instead of using frames directly, we decided to employ background-suppressed frames on the other stream. We used a simple method to estimate the backdrop to reduce computing costs. We start by calculating the average of each frame. Background information is mostly included in the average frame because it is constant between frames.



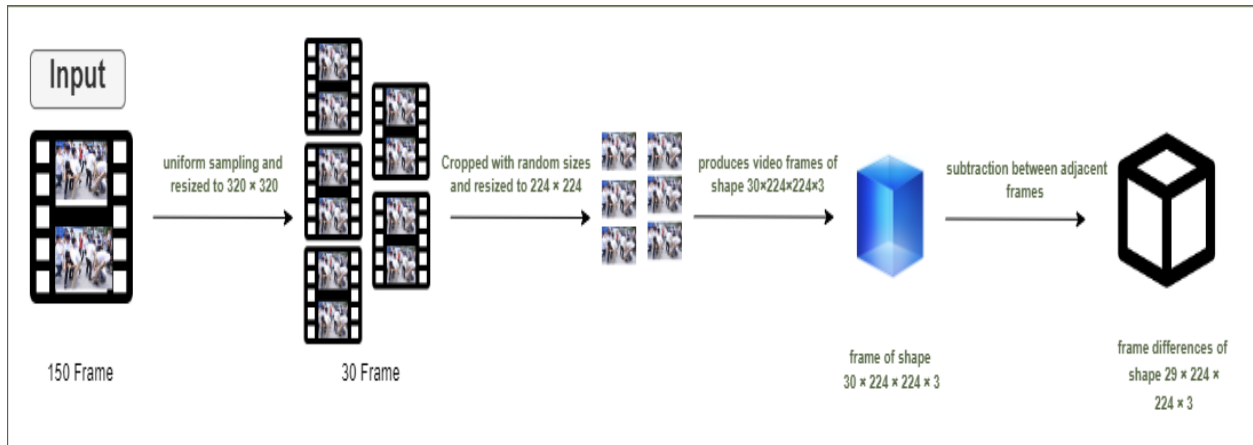
This average is subtracted from each frame so that the background data is muted and the moving objects in the frame are highlighted. As a result, the algorithm is encouraged to focus more on relevant data since aggressive actions are more frequently defined by body motions than by immobile backdrop features. Equation 5.2 serves as the formal representation of this procedure.

$$avg = \sum_{i=0}^N \frac{frame_i}{N}$$

$$bsf_i = |frame_i - avg| \quad (5.2)$$

The  $i$ th frame is denoted as  $frame_i$  the  $i$ th time-step of background suppressed frames that we input into our model( $bsf_i$ ), and the average of all the frames is termed average.

The impact of frame differences and background suppression on video frames is displayed in Figure 5.3. The primary focus of the frame difference is the shift in body postures, which provides temporal information like movements. However, background-suppressed frames preserve some textural or appearance-based information about the moving objects in the foreground even after suppressing the background pixels



**Figure 5. 4. Explains the stages of video processing before starting the training process**

#### 5.4.1.2 Training Methodology

In a video clip, there is usually duplicate information in adjacent frames. Thus, as in Figure 5.4, we used uniform sampling to retrieve only 32 frames for each movie, which we then scaled to 320 x 320. They were downsized to 224 x 224 and cropped to random sizes before placing them on the model. This results in 32 x 224 x 224 x 3 video frames. After subtracting the elements between successive frames, we obtained frame differences with dimensions of 31 x 224 x 224 x 3.

Memory limitations forced us to work in batches of only four. To reduce overfitting, a variety of data augmentation techniques were used during the training phase, including random brightness, random cropping, Gaussian blur, and random horizontal flips.

### 5.4.1.3 Implementation

The SepConvLSTM-C model, which was mentioned in the background section, was used and trained for approximately 120 epochs or until the model began to overfit. The CNNs were initialized using weights previously trained on the ImageNet dataset. Used the Xavier initialization [16] of the SepConvLSTM kernel. The hockey and movie datasets are very small, which can cause overfitting. Therefore, we first trained on the RWF-2000 dataset. Next, the weights of this trained model will start training on the other two datasets. To improve the model, we used the AMSGrad variant of the **Adam optimizer** [17]. Start training with a learning rate of  $4 \times 10^{-4}$ . After every 5 epochs, we halved the learning rate until it reached  $5 \times 10^{-5}$ . Keep it unchanged from that era. The model is optimized to minimize the sigmoid loss between the ground truth and the predicted label.

## 5.4.2 C3D

### 5.4.2.1 Data Preparation

We loaded videos to input into the model as a volume of batches with batch size equal to 8 using a data generator.

- Optical Flow

Optical flow is a task of per-pixel motion estimation between two consecutive frames in one video. The optical flow task implies the calculation of the shift vector for pixels as an object displacement difference between two neighboring images. The main idea of optical flow is to estimate the object's displacement vector caused by its motion or camera movements [18], as shown in Figure 5.6.

**Dense optical flow** computes the optical flow vector for every pixel of the frame, which may be responsible for its slow speed but leads to a more accurate result. It can be used for detecting motion in videos, video segmentation, and learning structure from motion. There are various implementations of dense optical flow, as shown in Figures 5.5 and 5.6. We will be using the Farneback method, one of the most popular implementations.





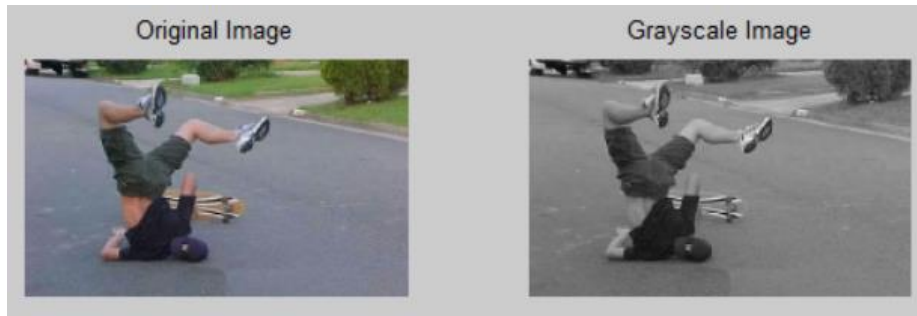
**Figure 5. 5 The dense optical flow algorithm output encoded as the HSV color scheme.**

In figure 5.5, We can convert the displacement coordinates  $(dx, dy)$  into polar coordinates as magnitude and angle for every pixel. Here we can encode angle and magnitude as Hue and Value respectively, while Saturation remains constant. To show the optical flow properly, we need to convert the HSV into BGR format.



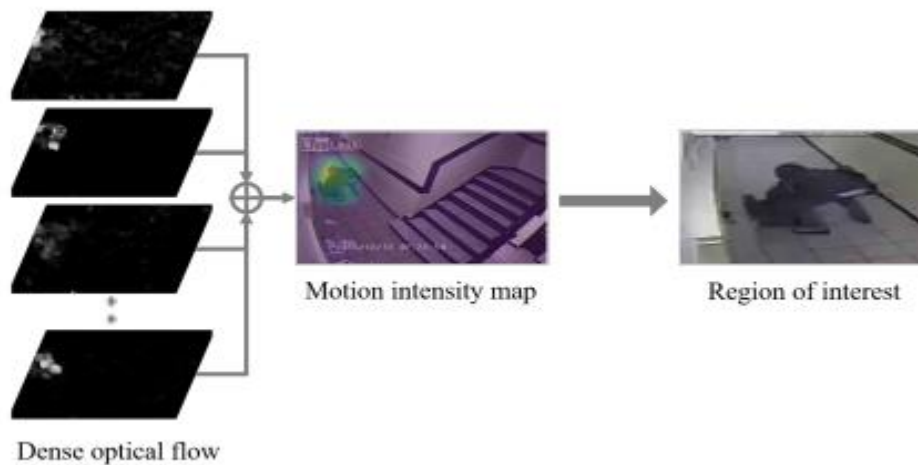
**Figure 5. 6. Result of optical flow after converting to HSV**

Before calculating optical flow, we converted RGB frames into gray frames, as shown in Figure 5.7.



**Figure 5. 7. Example of converting RGB image to gray image**

To achieve the desired effect, we resorted to calculating the dense optical flow using Gunner Farneback's method. The computed dense optical flow is a field of the 2-D displacement vector. Thus, we calculate the norm of each vector to obtain a heat map indicating the motion intensity. We use the sum of all the heat maps to calculate the basinal motion intensity map. The region of interest is extracted from the location with the most significant motion intensity [10], as shown in Figure 5.8.



**Figure 5. 8. Structure of the Flow Gated Network**

- Conversion to .npy files

After optical flow is applied to the original video files, they are then converted into (.npy) files, each of which is a tensor with shape [nb\_frames, img\_height, img\_width, 5]. The last channel contains 3 layers for RGB components and 2 layers for optical flows (a horizontal component and a vertical component), and this shape of tensor will be the input shape of the model.

### 5.4.2.2 Additional Preprocessing

- Uniform sampling

We sparsely sample frames from the video at a uniform interval for each input video and then generate a fixed-length video clip. By adopting both cropping and sampling, the amount of input data decreases significantly. The target length of video clips is 64, and the size of the cropped regions is  $224 \times 224$ . After sampling and cropping processes, the input data has a shape of  $(64 \times 224 \times 224 \times 5)$ .

- Data augmentation

To reproduce more real-world qualities to the clips, various data augmentation methods were applied to the training portion of the dataset. Color jitter, a type of data augmentation where the brightness, contrast and saturation are randomly changed, was applied to the clips to imitate real-world lightning conditions where cameras are placed in a dimly lit area or where the clips are overly exposed due to harsh lightning. Additionally, random flips, rotations and clipping were utilized to simulate different camera angles.

### 5.4.2.3 Implementation

The SGD optimizer was used with momentum = 0.9 and an initial learning rate of 0.01, with a learning rate decay of  $1e-6$ . Batch size was set to 8, and the model was trained for 30 epochs for all trials, with the architecture explained in the subsection above.

For our **first** trial, we used a slightly altered version of the architecture mentioned above, using both the optical flow and RGB channels as input. Initially, due to a lack of computational resources, we resorted to resizing the frames of the cropped regions from the video clips to  $150 \times 150$  instead of  $224 \times 224$ , to reduce the size of input and therefore save some resources. In addition to that, we added a batch normalization layer following every 2 convolution layers, as well as the L2 kernel regularizer, as further optimization.

For the **following** trial, after managing to acquire more powerful resources, we wanted to test how the model performs without all the elements that were added to the original architecture, to see if the addition of those layers could have negatively impacted the performance. The batch regularization layers and the kernel regularizer were removed, and the image resizing was restored back to the original  $224 \times 224$  once again. In this trial, our primary focus was achieving higher accuracy than in previous trials.

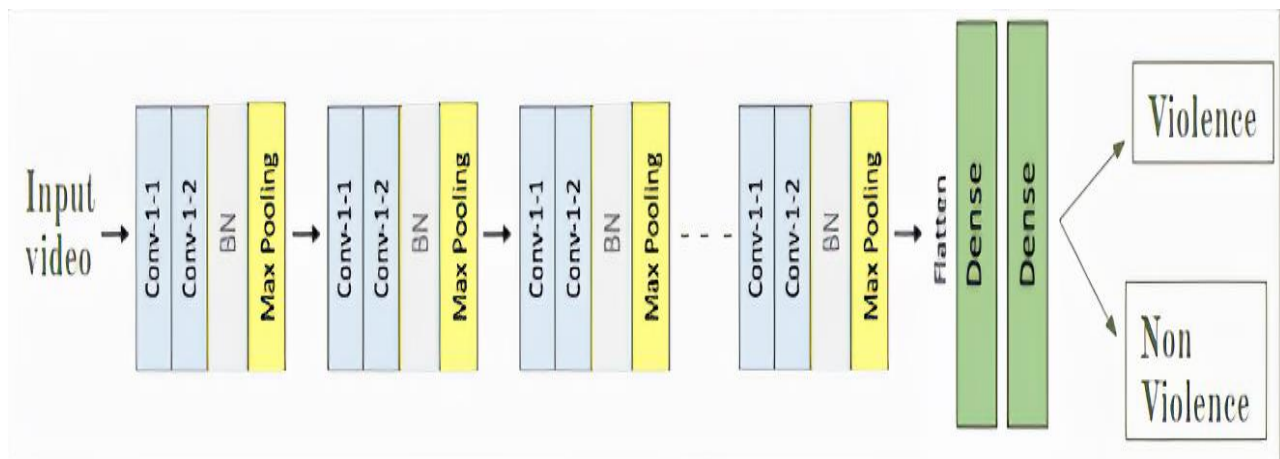
### 5.4.3 3DCNN

#### 5.4.3.1 Model Architecture

3DCNN refers to any convolutional neural network architecture that includes 3D convolutional layers. It is a more general term that encompasses various architectures designed for processing 3D data, including videos. While 3DCNNs can have different architectures and configurations, they all involve the use of 3D convolutions to capture spatiotemporal features.

3DCNNs can be trained using supervised learning with labeled data. However, they can also be trained using unsupervised or semi-supervised learning techniques, where the network learns from unlabeled or partially labeled data.

The structure of our proposed model with blocks of convolution layers followed by batch normalization layers, followed by pooling layers. Then the fully connected layers, as shown in Figure 5.9.



**Figure 5. 9. Architecture of the 3DCNN Model**

**Table 5. 1. The detailed parameters of the model structure**

Type	Filter Shape	Number Of Repeats
Conv3d Conv3d Batch Normalization MaxPool3d	1×3×3@16 3×1×1@16 none 1×2×2	2
Conv3d Conv3d Batch Normalization MaxPool3d	1×3×3@32 3×1×1@32 none 1×2×2	2
MaxPool3d	8×1×1	1
Conv3d Conv3d Batch Normalization MaxPool3d	1×3×3@64 3×1×1@16 none 1×2×2	2
Conv3d Conv3d Batch Normalization MaxPool3d	1×3×3@128 3×1×1@128 none 1×2×2	1
FC	128	1
FC	32	1
Sigmoid	1	1

### 5.4.3.2 Implementation

First, we loaded videos to input into the model as a volume of batches, and the batch size was set to 6 using a data generator, the model was trained for 40 epochs for all trials, in addition to resizing the frames of input videos to be  $160 \times 160$  and the input shape to be  $150 \times 160 \times 160 \times 3$ .

In the first trials, we used the architecture mentioned above in section 5.4.3.1 without batch normalization layer and regularization term, but in the last and final trials, we used them.

**Dropout layers** serve as a regularization technique to prevent overfitting by randomly selecting some neurons and making their weights equal to zero. The percentage of neurons to be dropped out is considered a hyperparameter.

**Batch normalization layers** help in making the model converge faster and can act as a regularization term to prevent overfitting by adding noise to the input. Batch normalization layers work by converting input to its z-score then scaling and shifting these values by learnable parameters before passing them to the activation function.

We used the sparse categorical cross-entropy loss function in most of the implementation trials. However, in the last trials, we used binary cross entropy as a loss function with the Adam optimizer, and the initial learning rate was set to 0.001.

We used a learning rate scheduler function that helps in managing the learning rate dynamically during training. By reducing the learning rate at regular intervals. Adapting the learning rate increases performance and reduces training time.

### 5.4.4 Transformer-based

We tried to use a pre-trained transformer model but due to a lack of resources, we couldn't reach appropriate results to discuss, but we built a model from scratch [19] and that is what we will discuss in detail.

#### 5.4.4.1 Data Preparation

First, we prepared video paths and their labels, and then we combined them into a data frame, in addition to some preprocessing techniques like cropping and padding shorter videos.

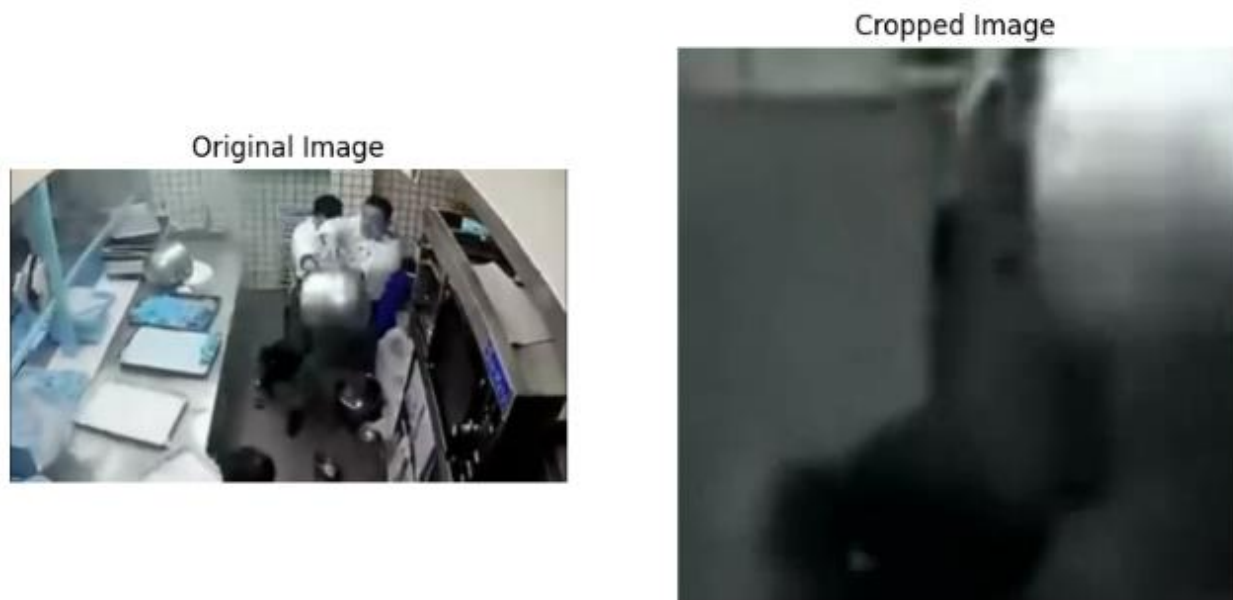
- Cropping

Crop the center of the original image based on the given height and width, which are variables in the overall trials as shown in Figure 5.10.



**Figure 5. 10. Example of center cropping of image**

But in other trials, we exchanged this technique with resizing because some videos lost their importance with it, as shown in figure 5.11.



**Figure 5. 11. An example of center cropping of image lost its importance**

- **Pad shorter**

Given a target length for all videos, which is also variable in all trials, ensure that all videos in a dataset have the same length.

#### 5.4.4.2 Model Architecture

For the model architecture, the transformer encoder was only used and the detailed layers of architecture as shown in Tale 2.2.

**Table 5. 2. The architecture layers of the transformer model.**

<b>Layer type</b>	<b>Output shape</b>
<b>Input</b>	<b>(None, max sequence frames, number of features)</b>
<b>Positional Embedding</b>	<b>(None, max sequence frames, number of features)</b>
<b>Transformer Encoder</b>	<b>(None, max sequence frames, number of features)</b>
<b>Global max pooling (1D)</b>	<b>(None, number of features)</b>
<b>Dropout</b>	<b>(None, number of features)</b>
<b>Dense (output)</b>	<b>(None, 1)</b>

#### 5.4.4.3 Implementation

We used pre-trained DenseNet-121 as mentioned in Chapter 2, as a feature extractor to extract a fixed number of features (1024) from a fixed length of frames in each video to input into the model as shown in Table 5.2 with shape (batch size, number of frames, number of features) for each video.

We have different trials with different hyperparameters that contain a fixed number of features extracted from each video, which equals 1024 features, the number of epochs, the image size, and the maximum sequence of frames in each video. We used binary cross entropy as a loss function with different optimizers.

In the first trial, the frame size was set to 128; the maximum sequence of frames was set to 30, and the model was trained for 20 epochs with an Adam optimizer.

In the second trial, the frame size was set to 256, the maximum sequence of frames was set to 70, and the model was trained for 50 epochs with the SGD optimizer, achieving higher accuracy.

In other trials, we also used different hyperparameters and tried to use different preprocessing techniques, like exchanging center cropping with resizing, as mentioned above, to achieve higher accuracy, but it resulted in lower test accuracy in addition to the model sometimes overfitting.



## Chapter 6. Experiments & Results

In this Chapter, we will discuss the experiments and the results of the models discussed in their implementation in Chapter 5 in detail. We experimented with several trials to reach the results. One of the issues we faced throughout these trials was that the results were not reproducible. We couldn't make it reproducible due to the dependence on GPUs. Working with GPUs and TensorFlow makes it difficult to produce reducible results, as the problem is that TensorFlow backends need to be adjusted. We were only able to set the seeds of Numpy, random, TensorFlow, and Sklearn. We mostly set the seed as 42, however, some experiments might not have used it and used the default seed instead. Trials had different settings, preprocessing pipelines, and sometimes model implementation modifications. We are going to discuss each trial and its results independently.

### 6.1 ConvLSTM model

First, we have the checkpoints trained on the movie, hockey, and RWF2000 datasets [10], as shown in Table 6.1. As we can see, Strategy C is the best in terms of results. We tried to fine-tune the checkpoint trained on the movie dataset. As we know, movies have more motion diversity than hockey datasets, so the results tested on the RWF2000 dataset were better.

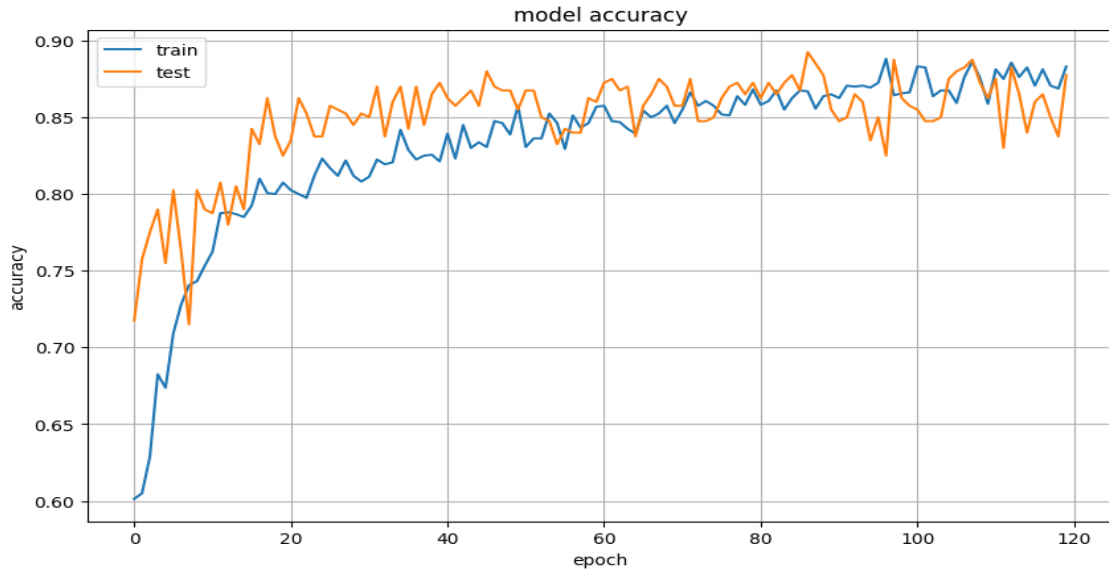
**Table 6. 1. Compare classification results for datasets**

Model Name	Movies	Hockey	RWF2000
SepConvLSTM-A	100%	99%	87.20%
SepConvLSTM-M	100%	99%	87.18%
SepConvLSTM-C	100%	<b><u>99.50%</u></b>	<b><u>89.25%</u></b>

Performance evaluation of the suggested approaches was conducted on 20% of the dataset, with the remaining 80% of the clips being utilized for training our models. The focus was on the RWF-2000 data set because it is more challenging than the other sets, and no work was done to improve its results much, as in the movie and hockey datasets, 100% was reached. As we saw previously, RWF-2000 consists of a set of live clips captured from surveillance cameras, unlike the hockey dataset, which consists of video clips, and focuses only on the violence prevalent in hockey matches, so the model trained on this group does not work well with the RWF-2000 dataset. As for the movie dataset, it consists of representative clips taken from different action and violence films, but in the end, they are unrealistic and small clips, so the model that was trained on them does not work well on RWF 2000 either. Even after working to improve as much as possible, its best result was only 73%, and that was the highest we achieved, as shown in Table 6.2

**Table 6. 2. Reported accuracy on the RWF2000 Test subset**

Model Name	Accuracy
SepConvLSTM-C (Pertained in Hockey)	56.40%
SepConvLSTM-C (Pertained in Movies)	68.99%
SepConvLSTM-C (Pertained in Movies & Fine Tuned on RWF)	<b><u>73%</u></b>



**Figure 6. 1. Pre-Trained SepCon LSTM-C model on RWF2000 train accuracy vs test accuracy**



Figure 6. 2. Pre-Trained SepCon LSTM-C model on RWF2000 train loss vs test loss

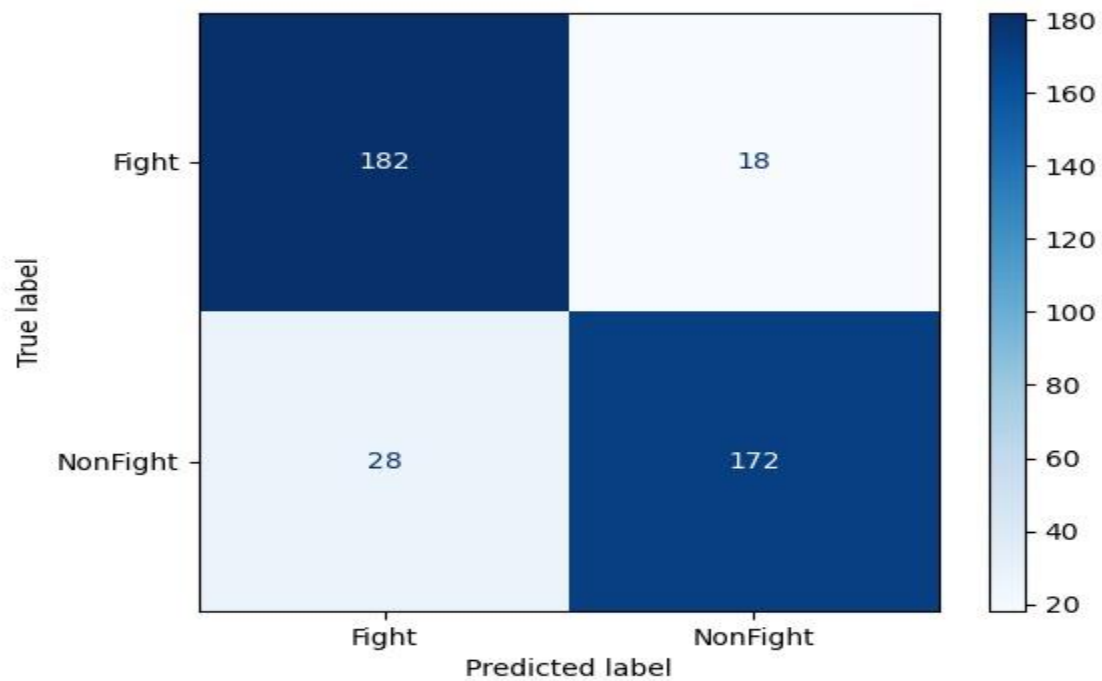


Figure 6. 3. Confusion matrix of the test set of the SepConvLSTM-C model

## 6.2 C3D model

We had 2 trials, as mentioned in the implementation section in Chapter 5, one without optimization techniques and the other with optimization techniques, and the results are shown in Table 6.3.

**Table 6. 3. Compare classification results for best trials**

Model	Train accuracy	Val accuracy	Test accuracy
Id_1(with)	87.81%	82.24%	72%
Id_2(without)	96.21%	84.24%	80.0%

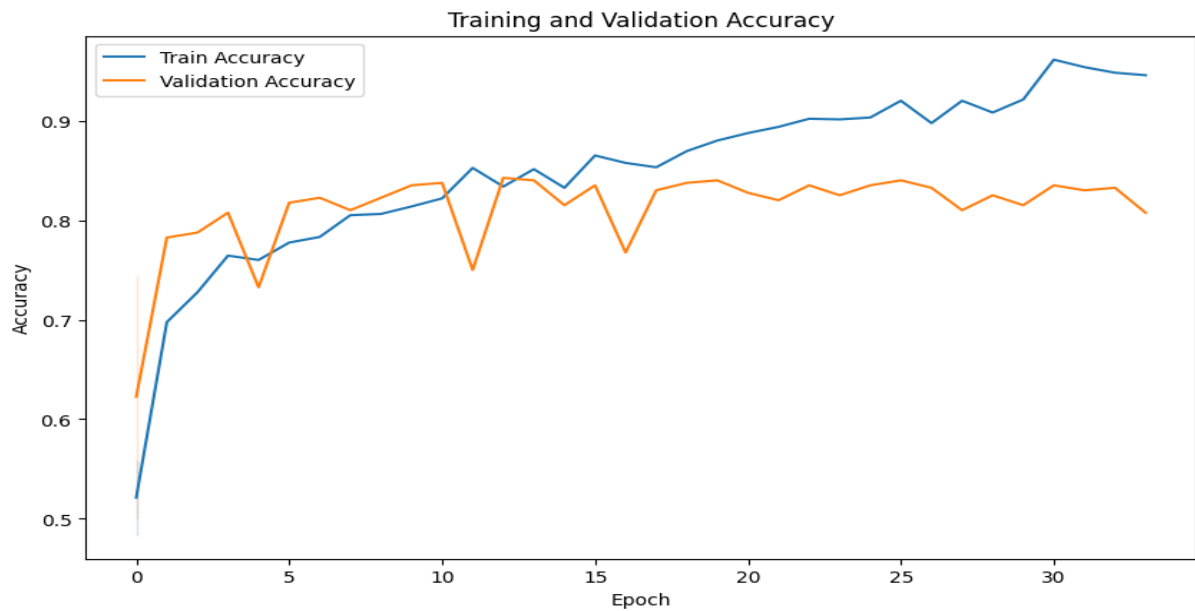


Figure 6. 4. Trail 2 of C3D model accuracy vs. validation accuracy.

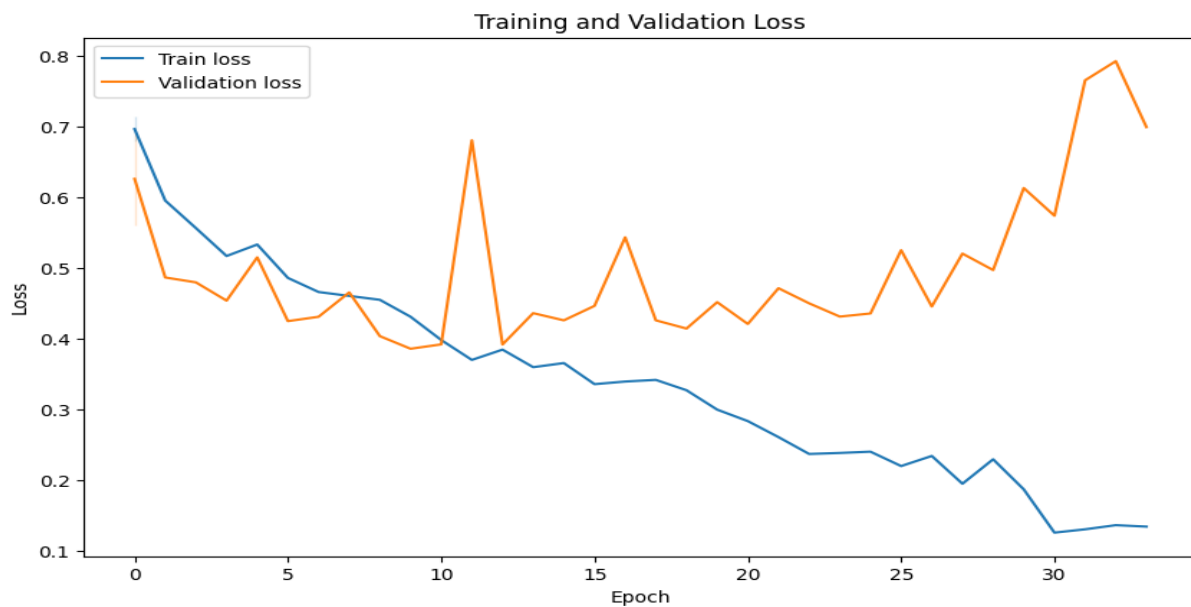


Figure 6. 5. Trail 2 of C3D model loss vs. validation loss.

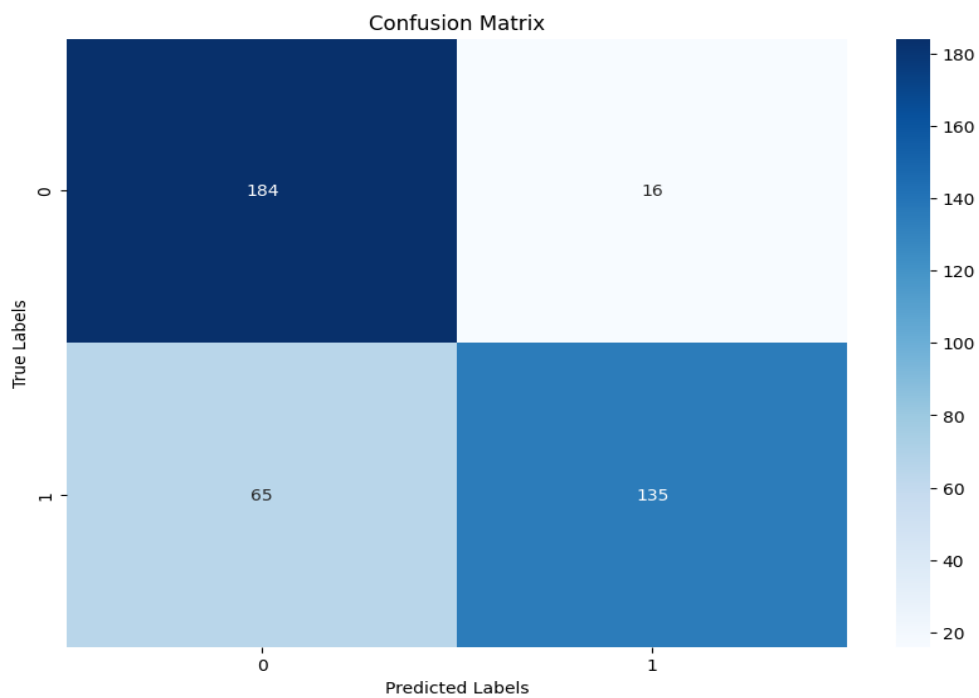


Figure 6. 6. Confusion matrix of the test set of C3D model trial 2

## 6.3 3DCNN model

We reached 78% higher test accuracy, as shown in the figure, and as we mentioned above, due to a lack of resources, we encountered a RAM crash in many situations if we wanted to increase the batch size as we used a data generator, as we mentioned in Chapter 5, where we could not use a batch size higher than 6.

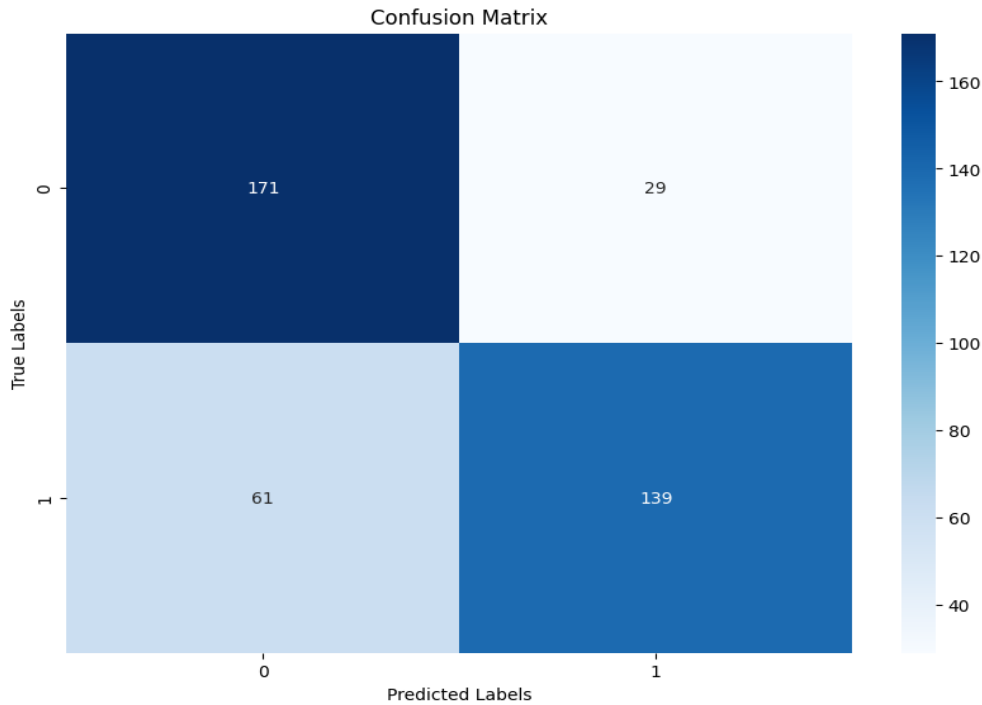


Figure 6. 7. Confusion matrix of a test set of 3DCNN model

## 6.4 Transformer-based model

We had different trials, as mentioned in Chapter 5 in detail and Table 6.4 shows the different results

Table 6. 4. Compare classification results for best trials

Model	Train accuracy	Val accuracy	Test accuracy
Id_1	93.76%	76.67%	72.75%
Id_2	88.25%	83.5%	79%

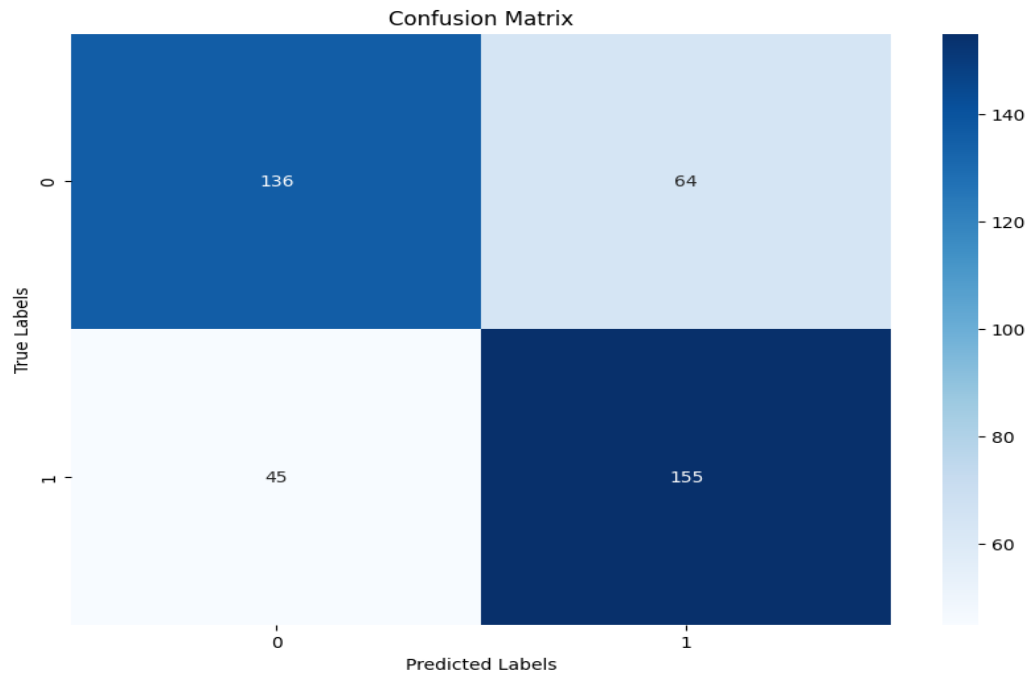


Figure 6. 8. Confusion matrix of a test set of transformer model trial 1

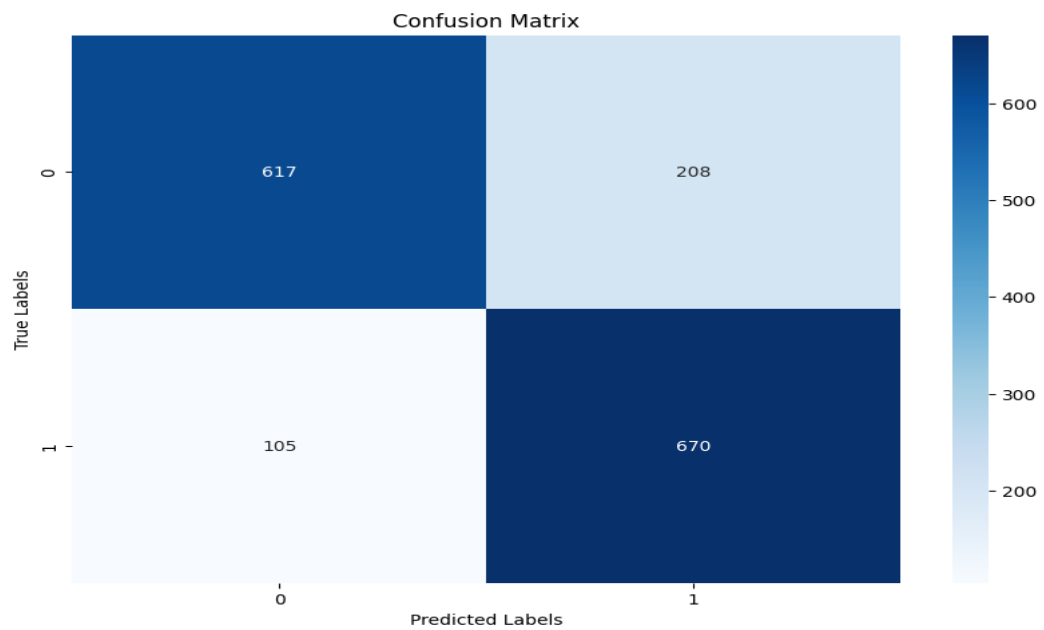


Figure 6. 9. Confusion matrix of train set of transformer model trial 1

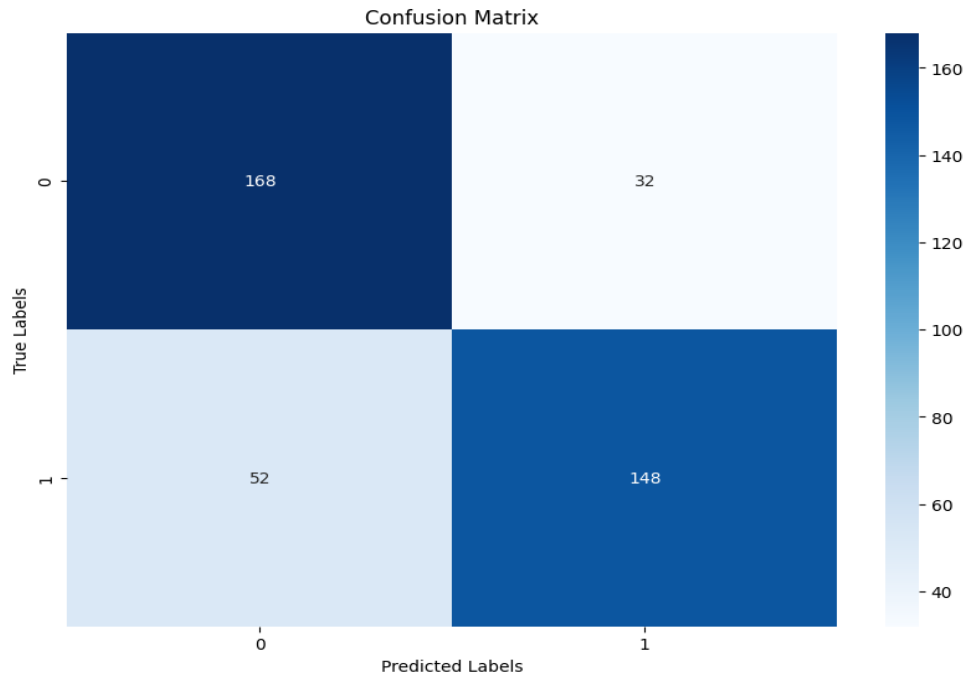


Figure 6. 10. Confusion matrix of test set of transformer model trial 2

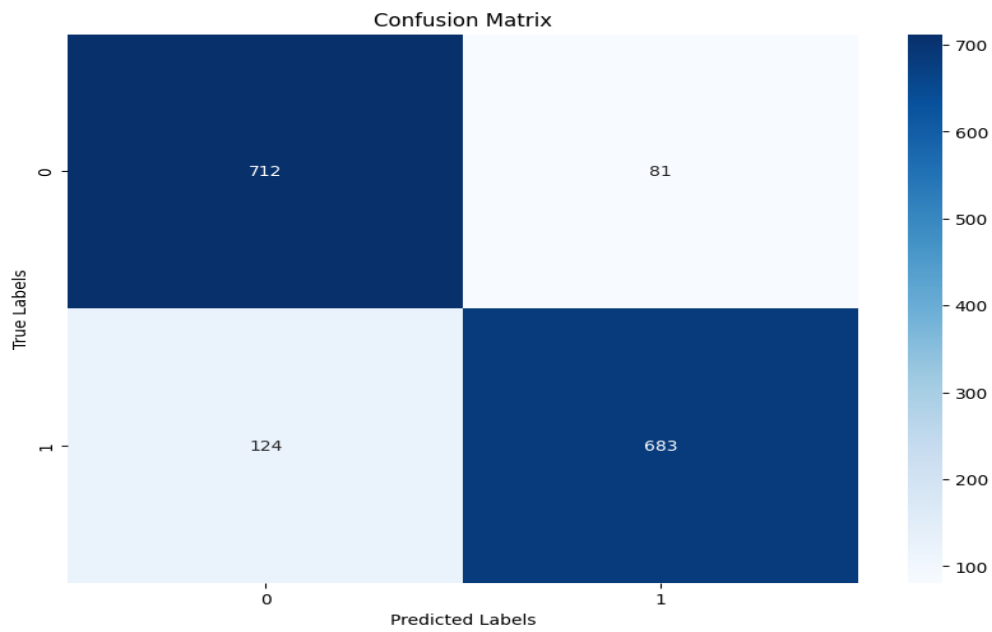


Figure 6. 11. Confusion matrix of train set of transformer model trial 2



## 6.5 Results Summary

As shown in Table 6.5, we combined classification reports for the best trials of each model.

**Table 6. 5. All results of all models for the RWF-2000 dataset**

	ConvLSTM			C3D			3DCNN			Transformer			support
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	
0	0.87	0.91	0.89	0.74	0.92	0.82	0.74	0.85	0.79	0.76	0.84	0.80	<b>200</b>
1	0.91	0.86	0.88	0.89	0.68	0.77	0.83	0.69	0.76	0.82	0.74	0.78	<b>200</b>
<b>Accuracy</b>	<b>0.89</b>			0.80			0.78			0.79			<b>400</b>
<b>Macro avg.</b>	0.89	0.89	0.88	0.82	0.80	0.79	0.78	0.77	0.77	0.79	0.79	0.79	<b>400</b>
<b>Weighted avg.</b>	0.89	0.89	0.88	0.82	0.80	0.79	0.78	0.78	0.77	0.79	0.79	0.79	<b>400</b>

## 6.6 Comparative Analysis

As shown in Table 6.5, we achieved high accuracy with the ConvLSTM model compared with other models, which is 87% test accuracy.

**Table 6.6. Comparison between related work and our work on the test subset of RWF-2000**

Method	Accuracy
<i>Related work</i>	
SepConvLSTM-C[6]	<b>89%</b>
C3D [6]	85.75%
P3D [6]	87.25%
3DCNN (RGB only) [6]	85%
<i>Our work</i>	
ConvLSTM	<b>89%</b>
C3D	80%
3DCNN	78%
Transformer	79%

Compared to the previous work, as shown in Table 6.6, we used the same ConvLSTM model and fine-tuned it, achieving the same result. The results of our work with the C3D model are different when we use the same architecture, train it from scratch with different preprocessing techniques, and make changes to the model's layers, like adding optimization techniques, but these changes lead to lower accuracy, and also when we use their checkpoint to evaluate the model due to lack of resources and GPUs, achieving 80% test accuracy and 84% validation accuracy.

Using the 3DCNN model like the C3D model architecture but not the same, as we explained in Chapter 5 in detail, the model deals with RGB channels only with different preprocessing techniques and data preparation, and we do not use optical flow channels, achieving 78% test accuracy.

With a lack of papers, articles, and resources for using the transformer model with our dataset, we could not find a benchmark for our results, so we used the transformer model and trained from scratch with different hyperparameters and preprocessing techniques, making changes to the architecture, and achieving different results.

## Chapter 7. Data Analysis

In our data analysis, our main objective was to identify and categorize the violent types within the RWF-20000 dataset.

First, we have extracted the meaningful features from our trained 3DCNN model by following this pipeline:

- We loaded video frames from the dataset using a specified path and resized them to a size of (150x160x160) to ensure uniformity of all frames.
- We fed the preprocessed video frames into the 3DCNN model.
- We obtained the outputs of each layer by creating functors, which are functions that take the model's input and return the desired layer outputs.
- We focused on the second-to-last layer's output (the layer before the classification layer) to use this output as our feature vector.

The feature vector we obtained had 32 dimensions, and the high dimensionality of the data made it difficult to cluster this data directly, so in our first experiment we performed a dimensionality reduction before applying the clustering algorithm.

### 7.1 Dimensionality Reduction

Dimensionality reduction techniques address this issue by transforming the data into a lower-dimensional representation while retaining important patterns and structures. We performed two techniques:

#### 7.1.1 t-SNE

The basic idea behind t-SNE is to convert similarities between data points in the high-dimensional space into probabilities, and then map these probabilities to a lower-dimensional space in a way that preserves the relationships between the data points as much as possible. The procedure for t-SNE involves the following steps:

- Compute pairwise similarities between data points in a high-dimensional space using a Gaussian kernel.
- Construct probability distributions based on the pairwise similarities, representing the relationships between data points.
- Initialize the positions of data points randomly in a low-dimensional space.
- Compute pairwise similarities between data points in the low-dimensional space using the student's t-distribution.
- Optimize the embedding by iteratively adjusting the positions of data points to minimize the difference between high-dimensional and low-dimensional pairwise similarities.

- Continue the optimization process until a stopping criterion is met.

### 7.1.2 UMAP

The basic idea behind UMAP is to construct a low-dimensional representation of the data that preserves the global and local structure of the high-dimensional space. UMAP uses a graph-based approach to build a topological representation of the data, which is then embedded in a low-dimensional space using stochastic gradient descent. The procedure for UMAP involves the following steps:

- Compute pairwise distances between data points in the high-dimensional space.
- Construct a fuzzy simplicial set to represent the local neighborhood structure.
- Optimize the low-dimensional embedding using stochastic gradient descent.
- Perform graph layout based on the low-dimensional embedding.
- Refine the embedding by adjusting the positions of the data points.
- Iterate the optimization and refinement steps until convergence.
- Visualize and analyze the final low-dimensional embedding.

After performing dimensionality reduction on the 32-dimensional feature vector, we obtained a lower-dimensional representation of the data. Then we applied clustering algorithms.

## 7.2 Clustering

It is the process of grouping similar data points based on their characteristics or patterns. It helps discover underlying structures, aids in data exploration and facilitates unsupervised learning. Clustering is used for preprocessing, anomaly detection, information retrieval, and more. It simplifies data analysis, enhances efficiency, and supports decision-making processes in various domains. We performed two techniques:

### 7.2.1 K-means

It is a centroid-based or partition-based clustering algorithm. This algorithm partitions all the points in the sample space into  $K$  groups of similarity. The similarity is usually measured using Euclidean distance. The algorithm is as follows (see Figure 7.1):

- $K$  centroids are randomly placed, one for each cluster.
- The distance of each point from each centroid is calculated.
- Each data point is assigned to its closest centroid, forming a cluster.
- The positions of the  $K$  centroids are recalculated.

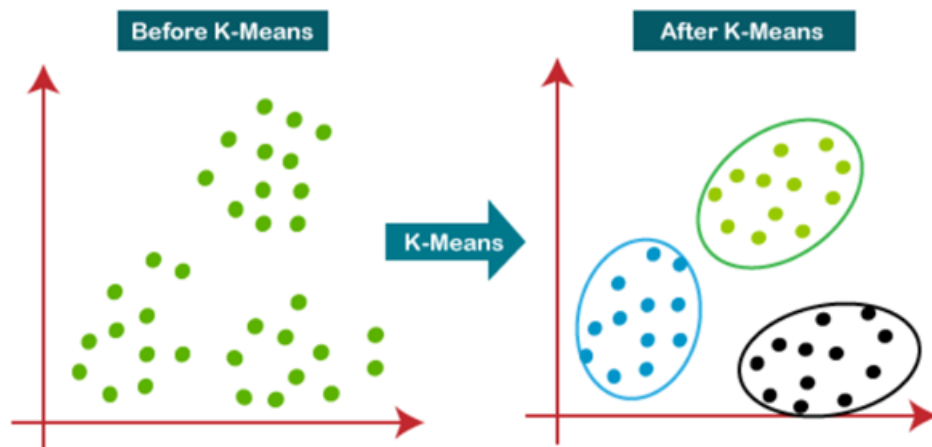


Figure 7. 1. Representation of how K-means works.

### 7.2.2 DBSCAN

It is a density-based clustering algorithm. This algorithm's key fact is that the neighborhood of each point in a cluster within a given radius ( $R$ ) must have a minimum number of points ( $M$ ). This algorithm has proved extremely efficient in detecting outliers and handling noise. The algorithm is as follows:

- The type of each point is determined. Each data point in our dataset may be either of the following (figure 7.2):
  - **Core Point:** A data point is a core point if there are at least  $M$  points in its neighborhood within the specified radius ( $R$ ).
  - **Border Point:** A data point is classified as a border point if:
    - Its neighborhood contains less than  $M$  data points, or
    - It is reachable from some core point, it is within  $R$ -distance from a core point.
  - **Outlier Point:** An outlier is a point that is not a core point and is also not close enough to be reachable from a core point.
- The outlier points are eliminated.
- Core points that are neighbors are connected and put in the same cluster.
- The border points are assigned to each cluster.

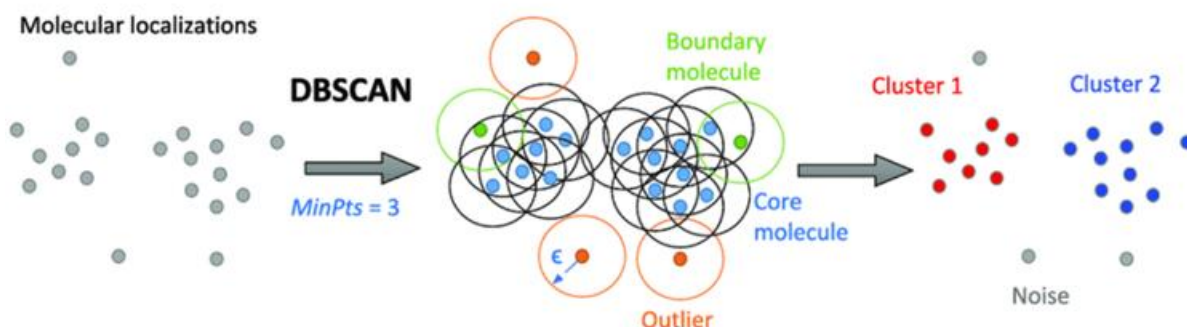


Figure 7. 2. Representation of how DBSCAN works

The outcomes obtained from applying the t-SNE dimensionality reduction technique with K-means clustering ( $k = 4$ ) (figure7.3) and the results of using UMAP dimensionality reduction technique with DBSCAN clustering ( $R = 0.8$ ,  $M = 10$ ) (figure 7.4).

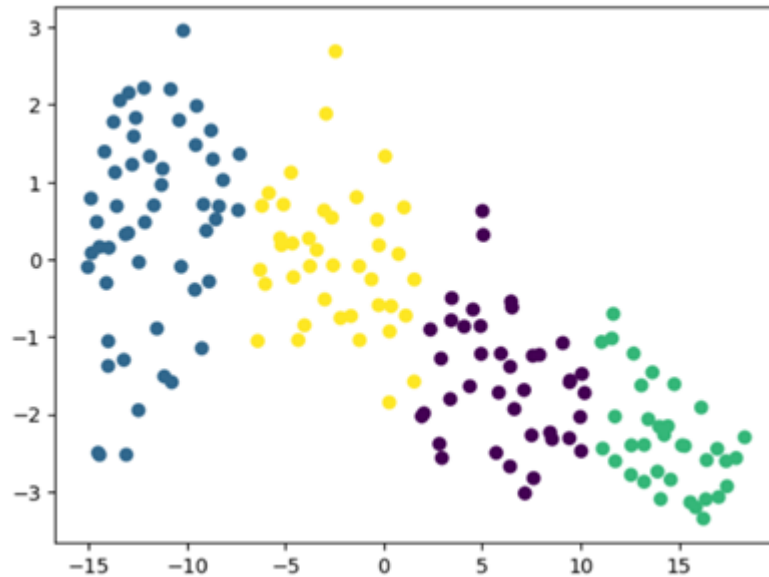


Figure 7. 3 t-SNE with K-means

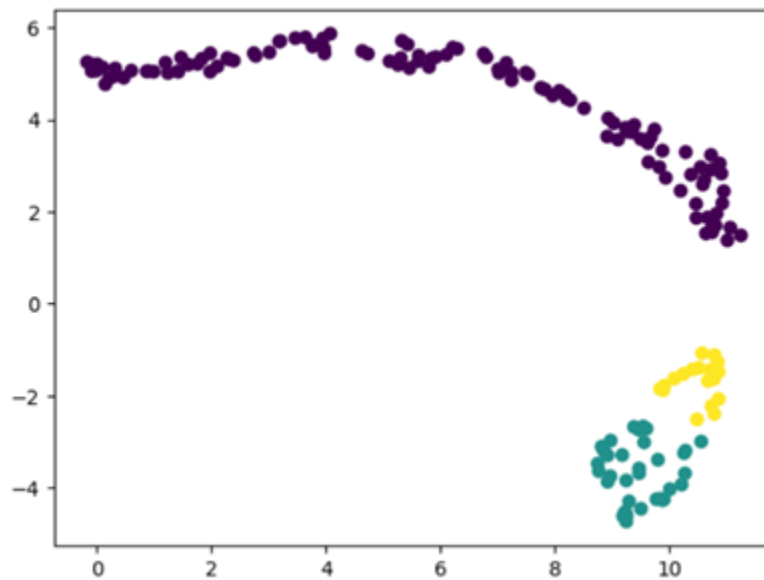


Figure 7. 4. UMAP with DBSCAN

These figures visually illustrate the clusters formed in the reduced-dimensional space.

To evaluate the results of this experiment, we conducted a two-person testing approach, where we carefully tested the clustering outcomes to identify the distinct groups formed by the clustering algorithm to assign the appropriate violence type to each identified group.

We encountered challenges in obtaining clear and meaningful results. We thought that the dimensionality reduction might have led to a loss of important information from the feature vector. So, we conducted a second experiment. In this experiment, we performed clustering directly on the original feature vector without applying dimensionality reduction, this allowed us to retain the complete set of features, and then applied dimensionality reduction techniques to visualize the clustering results in a more easily interpretable way. The outcomes obtained from applying k-means clustering ( $k = 4$ ) then reducing the dimensions by dimensionality reduction technique t-SNE (figure 7.5) and the outcomes obtained from applying DBSCAN clustering ( $R = 0.8$ ,  $M = 10$ ) then reducing the dimensions by dimensionality reduction technique t-SNE (figure 7.6).

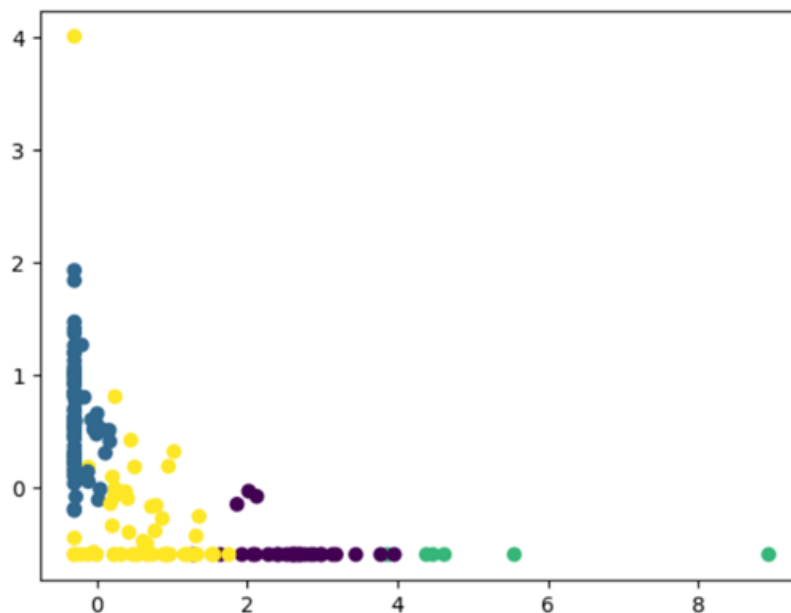


Figure 7. 5. K-means then applying t-SNE.

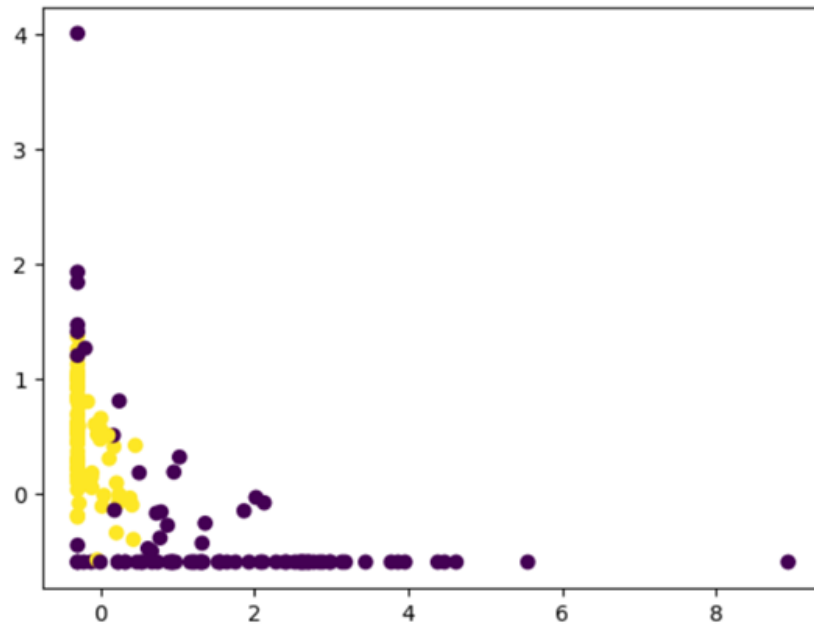


Figure 7. 6. DBSCAN then applies t-SNE.

We also evaluated this experiment by a two-person testing approach, In the case of DBSCAN clustering, which does not require a predefined number of clusters, the results indicated that the data was grouped into two clusters. However, with careful examination, we found that these clusters did not match with the known types of violence that we are familiar with.



## Chapter 8. User Manual

### 8.1 Detailed Features – Portal User Flow

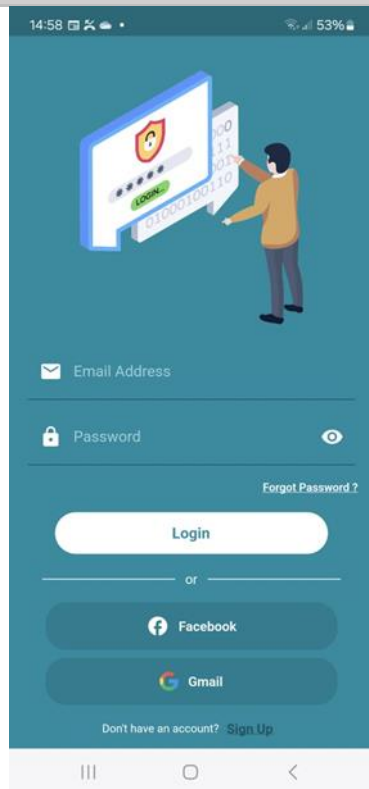
#### 8.1.1 Login

Portal user “Everyone eligible to use the application “.

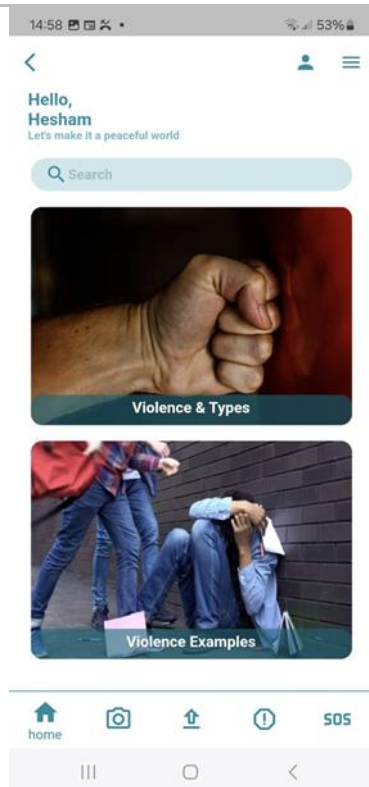
Feature no. 1		
Portal user Story	As a Portal user, I want to log in to access the application.	
Portal user flow	<ul style="list-style-type: none"><li>• Portal users will go through the application.</li><li>• Portal user will enter the credentials:</li><li>• Email Address</li><li>• Password</li><li>• Portal users will be navigated to the home page.</li></ul>	
Happy Scenario	<ul style="list-style-type: none"><li>• Portal user will login successfully with valid credentials.</li></ul>	
Fields	Name	Description
	Email address	<ul style="list-style-type: none"><li>• Mandatory</li><li>• Email format</li></ul>
	Password	<ul style="list-style-type: none"><li>• Mandatory</li><li>• No spaces</li></ul>
Exceptions – negative Scenarios	Action	Message
	Email address empty	This field is required.
	Email address is not in email format	Please enter a valid email address.
	Password empty	This field is required.
	Wrong password	The email or password is incorrect

## Steps to do

### Login



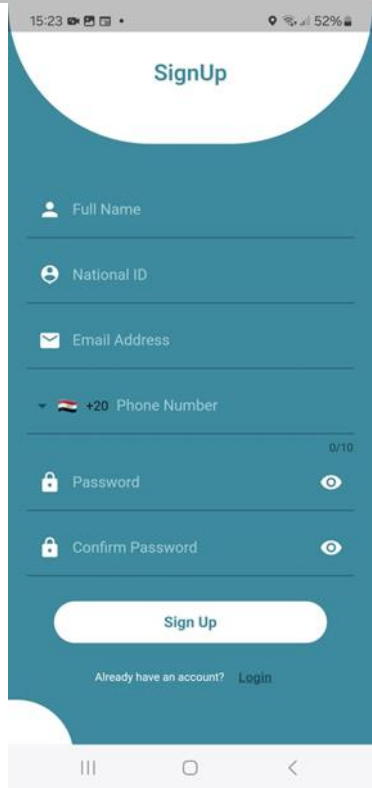
### Success login.

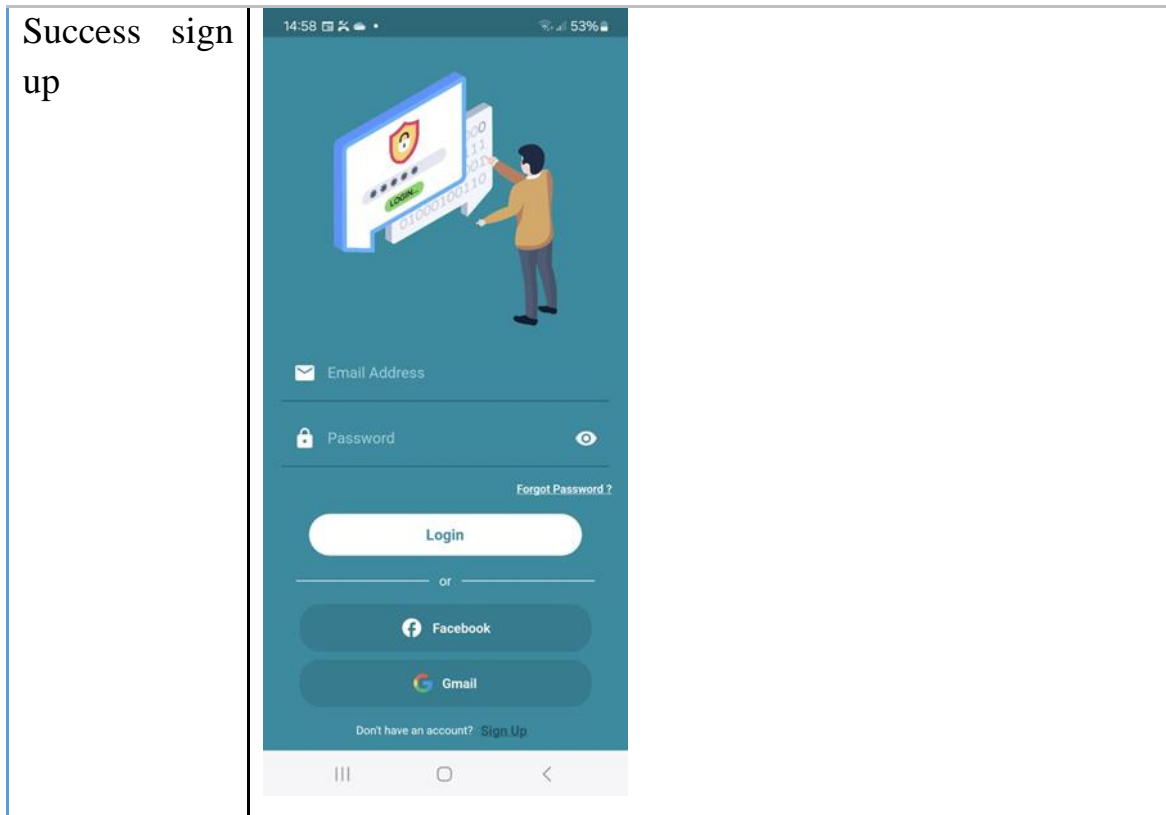


## 8.1.2 Sign up

Feature no. 2		
Portal Story	user	As a Portal user I want to create an account in the application.
Portal flow	user	<ul style="list-style-type: none"><li>• Portal users will go through the application.</li><li>• Portal users will press “Sign up”</li><li>• Portal user will enter the credentials:<ul style="list-style-type: none"><li>• Name</li><li>• National ID</li><li>• Email Address</li><li>• Phone Number</li><li>• Password</li><li>• Confirm Password</li></ul></li><li>• Portal user will be navigated to the login page.</li></ul>
Happy Scenario		<ul style="list-style-type: none"><li>• A new account will be created successfully</li><li>• User will be navigated to the login page.</li></ul>
Fields	Name	Description
	Full Name	<ul style="list-style-type: none"><li>• Mandatory</li><li>• Letters Only</li></ul>
	National ID	<ul style="list-style-type: none"><li>• Mandatory</li><li>• Numbers Only</li></ul>
	Email address	<ul style="list-style-type: none"><li>• Mandatory</li><li>• Email format</li></ul>
	Phone Number	<ul style="list-style-type: none"><li>• Mandatory</li><li>• Choose the country code</li><li>• Numbers Only</li></ul>
	Password	<ul style="list-style-type: none"><li>• Mandatory</li><li>• No spaces</li></ul>
	Confirm Password	<ul style="list-style-type: none"><li>• Mandatory</li><li>• No spaces</li><li>• Must be identical with password</li></ul>

Exceptions Scenarios	—	negative	Action	Message
			Full Name empty	This field is required.
			National ID empty	This field is required.
			Email address empty	This field is required.
			Phone number empty	This field is required.
			Incomplete phone number	Invalid phone number.
			Password empty	This field is required.

Steps to do	
Sign up	

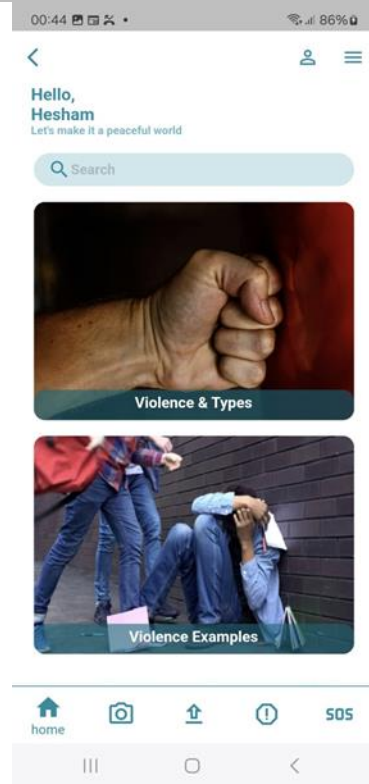


### 8.1.3 Delete account

Feature no. 3	
Portal user Story	As a Portal user I want to delete my account.
Portal user flow	<ul style="list-style-type: none"> <li>• Portal user will go through the application.</li> <li>• Portal user will enter the valid credentials and press Login.</li> <li>• Portal user will be navigated to the home page.</li> <li>• Portal user will press the profile icon from the app bar.</li> <li>• Portal user will be navigated to profile screen.</li> <li>• Portal user will press "Delete".</li> </ul>
Happy Scenario	<ul style="list-style-type: none"> <li>• Account will be deleted successfully.</li> </ul>

## Steps to do

Press profile icon

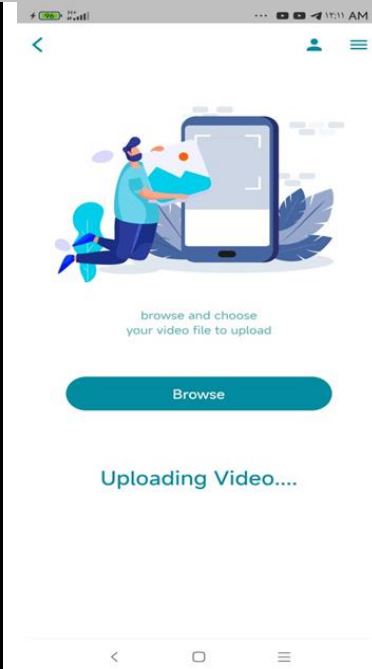


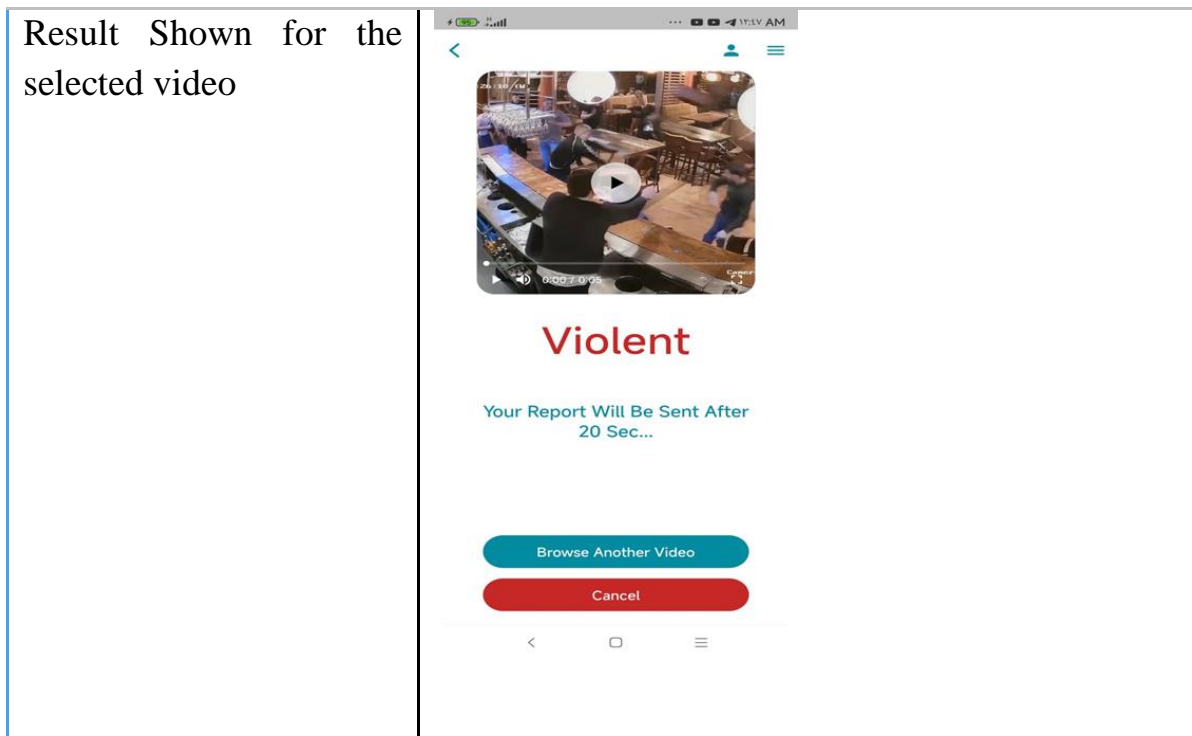
Press "Delete"



## 8.2 Main Features

### 8.2.1 Upload video

Feature no. 4	
Portal user Story	As a Portal user I want to detect violence in a video.
Portal user flow	<ul style="list-style-type: none"><li>• Portal user will go through the application.</li><li>• Portal user will enter the valid credentials and press Login.</li><li>• Portal user will be navigated to the home page.</li><li>• Portal user will press upload video icon from the bottom navigation bar.</li><li>• Portal user will pick a video from the gallery.</li></ul>
Happy Scenario	<ul style="list-style-type: none"><li>• Detection will be done successfully.</li><li>• The result will be shown.</li><li>• An email will be sent to report on this violence.</li><li>• User can upload another video or cancel.</li></ul>
Steps to do	
Press upload video icon and choose a video.	



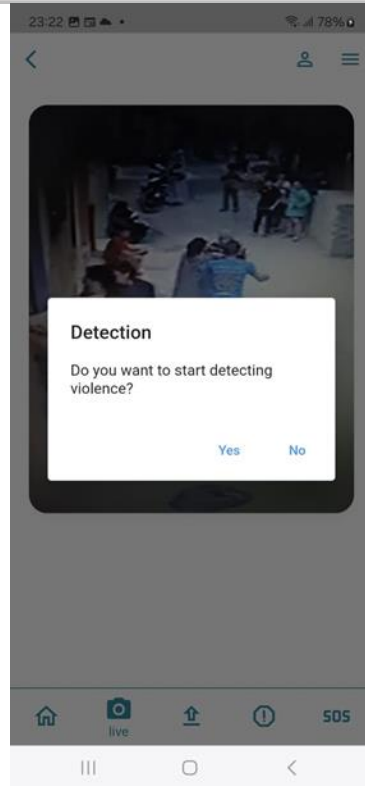
## 8.2.2 Live detection

Feature no. 5	
Portal user Story	As a Portal user I want to detect violence using live stream.
Portal user flow	<ul style="list-style-type: none"> <li>• Portal users will go through the application.</li> <li>• Portal user will enter the valid credentials and press Login</li> <li>• Portal user will be navigated to the home page.</li> <li>• Portal user will press the camera icon in the bottom navigation bar.</li> <li>• Portal user will press 'yes' to start detecting.</li> </ul>
Happy Scenario	<ul style="list-style-type: none"> <li>• Live violence detection will be done successfully.</li> <li>• Portal user will be navigated to violent clip screen.</li> <li>• The violent clip will be played.</li> <li>• Portal user can press "Report" to report on this violence.</li> <li>• An Email will be sent to the admin about this detected violence, it also will contain the category of the violence.</li> </ul>

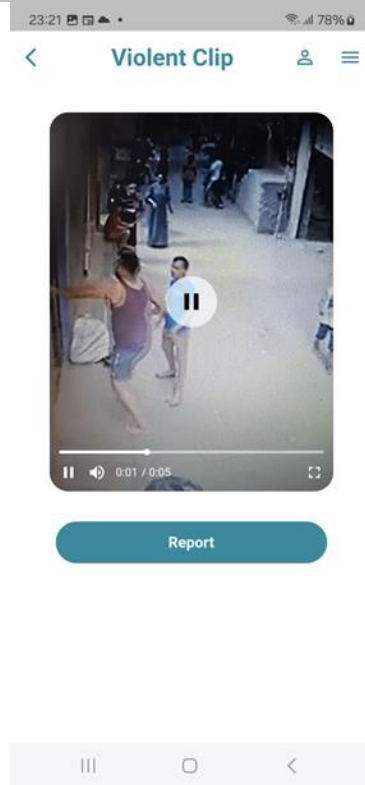


## Steps to do

Press camera icon,  
press yes

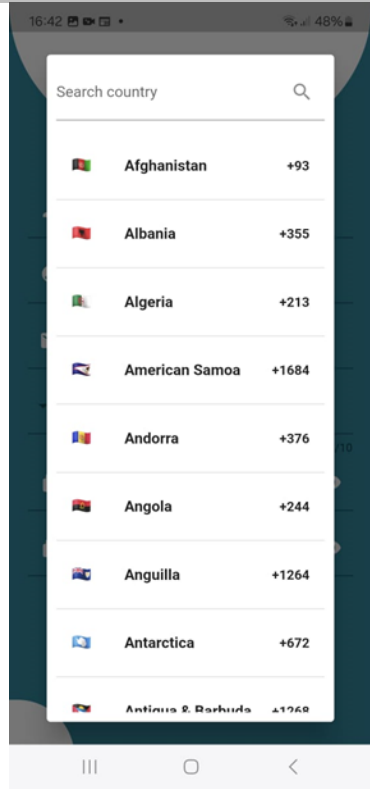


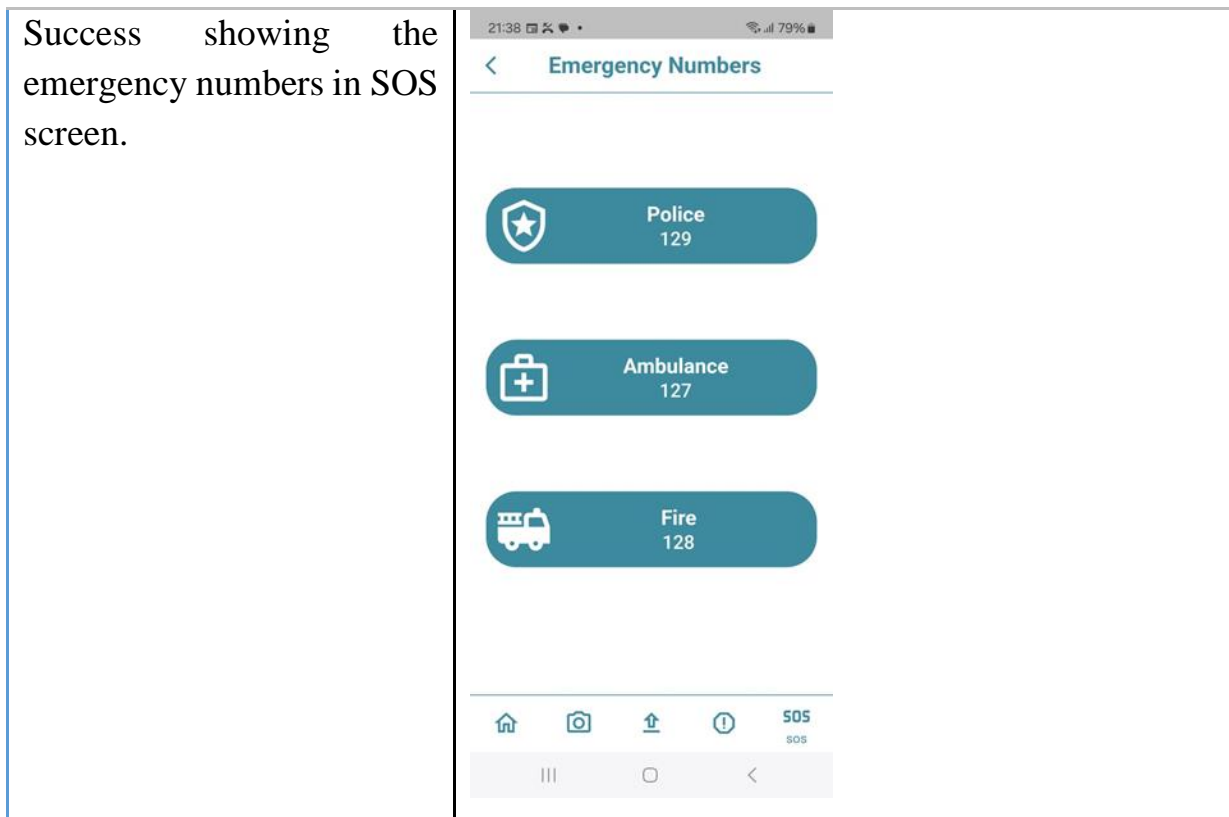
Success live  
violence detection.



### 8.2.3 SOS

Feature no. 6		
Portal user Story	user	As a Portal user I want to know the emergency numbers in my country.
Portal user flow	user	<ul style="list-style-type: none"><li>• Portal users will go through the application.</li><li>• Portal user will enter the valid credentials and press Login.</li><li>• Portal user will be navigated to the home page.</li><li>• Portal user will press the SOS icon in the bottom navigation bar.</li></ul>
Happy Scenario		<ul style="list-style-type: none"><li>• Emergency numbers in a user country will be shown.</li></ul>

Steps to do	
When selecting the country in signup	



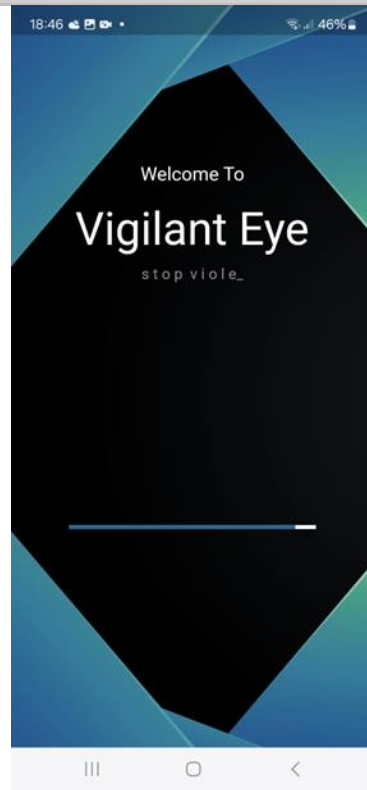
## 8.3 Side Features

### 8.3.1 Violence declaration

Feature no. 7		
Portal user Story	user	As a Portal user I want to know more about violence.
Portal user flow	user	<ul style="list-style-type: none"> <li>Portal user will open the application.</li> </ul>
Happy Scenario		<ul style="list-style-type: none"> <li>Information about violence, its statistics and how to stop violence.</li> </ul>

## Steps to do

When open the application



Success showing violence declaration



Success showing violence statistics



Success showing violence stopping



## Chapter 9. Conclusion and Future Work

### 9.1 Conclusion

The goal of this project was to provide an easy way to detect acts of violence that are currently widespread, to help government institutions such as schools or people such as parents in Egypt detect acts of violence faster before the problem occurs, and to take necessary action procedures from the competent authorities.

We applied innovative, cutting-edge methods that have been discovered over the past two years and are trying to improve, such as SepConvLstm, where we used a powerful new method to detect violent activity in real-world surveillance footage as RWF2000 Dataset. The proposed network can learn discriminative spatiotemporal features effectively which is reflected in the high recognition accuracy on standard datasets. Moreover, it is computationally efficient, making it suitable for deployment in time-sensitive applications and low-end devices. We have shown that the SepConvLSTM cell is a compact and robust alternative to the ConvLSTM cell. Since SepConvLSTM uses fewer parameters, stacking multiple layers of LSTM with the remaining connections seems feasible and may improve the results further.

C3D and 3DCNN are both types of convolutional neural networks designed to process spatiotemporal data (data that includes both spatial and temporal components, meaning that the data contains information about the location (spatial) and time (temporal) when an event or observation occurs), such as video clips. While C3D specializes in action recognition tasks, 3DCNNs can be applied to a wide range of spatiotemporal data analysis tasks. We also experimented with transform encoders used in many tasks involving sequential data, such as NLP and time series analysis. Key components include multi-head self-attention, feedforward neural network, and class normalization which may help in violence detection.

We performed data analysis to identify and categorize the violent types within RWF-20000 dataset. We extracted meaningful features from a trained 3D Convolutional Neural Network (3DCNN) model to obtain a feature vector. We performed dimensionality reduction using techniques such as t-SNE and UMAP, then applied clustering algorithms, including K-means and DBSCAN. However, with careful examination, we found that these clusters did not match with the known types of violence that we are familiar with.

## 9.2 Future Work

To improve the models' performance, creating a new dataset that includes videos of violence from different cameras, settings, and visual modes would improve prediction accuracy. Interpreting the model on a global scale would help build more accurate models and understanding of violence. In addition, working on developing multimodal models that combine data from different modalities, such as audio in videos, is an experiment that may lead to better predictive results. Collecting and using larger data sets from real Egyptian surveillance cameras would contribute to more effective violence detection. And trying to develop the methods that were used to distinguish between types of violence and trying to reach actual test groups that were eventually classified. The project has practical applications and aims to help institutions and people detect acts of violence faster to increase the speed of response before the situation escalates. As a team we plan to publish our app on the store. To help spread peace and reduce violence.

## References

- [1] P. Sernani, N. Falcionelli, S. Tomassini, P. Contardo, and A. F. Dragoni, “Deep Learning for Automatic Violence Detection: Tests on the AIRTLab Dataset,” *IEEE Access*, vol. 9, pp. 160580–160595, 2021, doi: 10.1109/ACCESS.2021.3131315.
- [2] United Nations Office on Drugs and Crime, *Global Study on Homicide 2023*. United Nations, 2023. doi: 10.18356/9789213587386.
- [3] E. L. Piza, B. C. Welsh, D. P. Farrington, and A. L. Thomas, “CCTV surveillance for crime prevention,” *Criminol Public Policy*, vol. 18, no. 1, pp. 135–159, Feb. 2019, doi: 10.1111/1745-9133.12419.
- [4] R. Vijeikis, V. Raudonis, and G. Dervinis, “Efficient Violence Detection in Surveillance,” *Sensors*, vol. 22, no. 6, Mar. 2022, doi: 10.3390/s22062216.
- [5] G. Garcia-Cobo and J. C. SanMiguel, “Human skeletons and change detection for efficient violence detection in surveillance videos,” *Computer Vision and Image Understanding*, vol. 233, p. 103739, Aug. 2023, doi: 10.1016/J.CVIU.2023.103739.
- [6] Z. Islam, M. Rukonuzzaman, R. Ahmed, Md. H. Kabir, and M. Farazi, “Efficient Two-Stream Network for Violence Detection Using Separable Convolutional LSTM,” Feb. 2021, doi: 10.1109/IJCNN52387.2021.9534280.
- [7] M. Sharma and R. Baghel, “Video Surveillance for Violence Detection Using Deep Learning,” 2020, pp. 411–420. doi: 10.1007/978-981-15-0978-0\_40.
- [8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks.” [Online]. Available: <http://code.google.com/p/cuda-convnet/>
- [9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2018, pp. 4510–4520. doi: 10.1109/CVPR.2018.00474.
- [10] M. Cheng, K. Cai, and M. Li, “RWF-2000: An Open Large Scale Video Database for Violence Detection,” Nov. 2019.
- [11] E. B. Nievas, O. Deniz Suarez, G. Bueno García, and R. Sukthankar, “Violence Detection in Video Using Computer Vision Techniques.” [Online]. Available: <http://visilab.etsii.uclm.es/>
- [12] T. Hassner, Y. Itcher, and O. Kliper-Gross, “Violent flows: Real-time detection of violent crowd behavior,” in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, IEEE, Jun. 2012, pp. 1–6. doi: 10.1109/CVPRW.2012.6239348.



- [13] W. Sultani, C. Chen, and M. Shah, “Real-World Anomaly Detection in Surveillance Videos,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2018, pp. 6479–6488. doi: 10.1109/CVPR.2018.00678.
- [14] S. Sudhakaran and O. Lanz, “Learning to Detect Violent Videos using Convolutional Long Short-Term Memory,” Sep. 2017.
- [15] A. Hanson, “Bidirectional Convolutional LSTM for the Detection of Violence in Videos.”
- [16] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks.” [Online]. Available: <http://www.iro.umontreal>.
- [17] S. J. Reddi, S. Kale, and S. Kumar, “On the Convergence of Adam and Beyond,” Apr. 2019.
- [18] Maxim Kuklin, “Optical Flow in OpenCV (C++/Python).”
- [19] A. Vaswani *et al.*, “Attention Is All You Need,” Jun. 2017.