UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

21 | 22

Degree work

Automatic violence detection in surveillance videos using deep learning

Guillermo García Cobo

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Degree as Computer Science

DEGREE WORK

Automatic violence detection in surveillance videos using deep learning

Author: Guillermo García Cobo
Advisor: Juan Carlos San Miguel Avedillo

mayo 2022

Francisco Tomás y Valiente, n^o 1
Madrid, 28049
Spain

Guillermo García Cobo
Automatic violence detection in surveillance videos using deep learning

Guillermo García Cobo
C\ Francisco Tomás y Valiente Nº 11

PRINTED IN SPAIN

Abstract

We live in a world in which we are increasingly being monitored by surveillance cameras. Their purpose is to reduce crime and violence in public places, mainly acting as a deterrent. However, we are not exploiting their true potential, as they are used after, to provide evidence, and not before or during the crime. Being able to change the action time would have a great positive impact on those negatively affected by crime, reducing or even preventing their suffering. To do so, we would need to lively monitor all available cameras, which is simply impossible. However, new advances in the image processing field, especially motivated by the irruption of deep learning, provide tools to build solutions that could take advantage of the huge amount of available videos and automatically detect crimes in them.

In this Final Degree Thesis (Trabajo Fin de Grado), we propose a novel deep learning architecture that is able to accurately and efficiently detect violent crimes in surveillance videos. To achieve this ambitious goal, we rely on what we believe are the most essential pieces of information to detect violence, namely: human bodies and their movement. To this end, we have used human pose extractors and motion estimators as the input of our proposal. Afterward, we combine them using a novel method of our own, which performs significantly better than other combination alternatives of the literature. Finally, to account for both spatial and temporal information, we use a convolutional alternative of the standard LSTM, a ConvLSTM. Moreover, to use this layer efficiently, we have solved a bad memory management of the original implementation by Keras, which has resulted in a merged pull request to the open-source library.

Once the architecture was built, we validated our proposal on a public benchmark dataset, specific for violence detection in surveillance videos. The experiments carried out help us to understand the potential and the contribution of each component of our architecture. Moreover, they demonstrate the efficacy and efficiency of the proposal as a whole, since we surpass state-of-the-art results with much fewer parameters.

Keywords

Violence detection, computer vision, pose estimation, deep learning, ConvLSTM

Resumen

Vivimos en un mundo en el que estamos cada vez más vigilados por cámaras de seguridad. Su propósito es reducir el crimen y la violencia en lugares públicos, actuando principalmente como elemento disuasorio. Sin embargo, no estamos explotando su verdadero potencial, ya que se utilizan después, para proporcionar pruebas, y no antes o durante los crímenes. Poder cambiar el tiempo de acción tendría un gran impacto positivo en las víctimas de crímenes, reduciendo o incluso previniendo su sufrimiento. Para hacerlo, necesitaríamos monitorear en vivo todas las cámaras disponibles, lo cual es simplemente imposible. Sin embargo, los nuevos avances en el campo del procesamiento de imágenes, especialmente motivados por la irrupción del aprendizaje profundo, brindan herramientas para construir soluciones que podrían aprovechar la gran cantidad de videos disponibles y detectar automáticamente delitos en ellos.

En este Trabajo Fin de Grado proponemos una novedosa arquitectura de aprendizaje profundo que es capaz de detectar de manera precisa y eficiente violencia en videos de vigilancia. Para lograr este ambicioso objetivo, nos basamos en lo que creemos que son las piezas de información esenciales para detectar violencia, a saber: los cuerpos humanos y su movimiento. Con este fin, hemos utilizado extractores de pose y estimadores de movimiento como entrada de nuestra propuesta. Posteriormente, los combinamos utilizando un método novedoso propio, que funciona significativamente mejor que otras alternativas de combinación de la literatura. Finalmente, para tener en cuenta tanto la información espacial como la temporal, usamos una alternativa convolucional de la LSTM estándar, una ConvLSTM. Además, para usar esta capa de manera eficiente, hemos resuelto una mala gestión de la memoria de la implementación original de Keras, lo que resultó en una contribución a la librería de código abierto.

Una vez construida la arquitectura, validamos nuestra propuesta sobre un conjunto de datos público de referencia, específico para la detección de violencia en videos de vigilancia. Los experimentos realizados nos ayudan a comprender el potencial y la contribución de cada uno de los componentes de nuestra arquitectura. Además, demuestran la eficacia y eficiencia de la propuesta en su conjunto, ya que superamos los resultados del estado del arte con muchos menos parámetros.

Palabras clave

Detección de violencia, visión por computador, estimación de pose, aprendizaje profundo, ConvLSTM

Table of Contents

Lists

List of figures

List of tables

1 Introduction

1.1. Motivation

In the last years, urban violence and crime have posed an increasing threat to society. To give a sense of the magnitude, we resort to data. According to the *Global Study on Homicide* [1], elaborated by the United Nations (UN), almost half a million people were killed in homicides worldwide in 2017. This surpasses by far the 89,000 killed in armed conflicts and the 19,000 killed in terrorist attacks. Moreover, the report states that there is a clear uptrend in the number of homicide victims over the last years.

One of the recurrent policies adopted by local authorities to reduce crime has been installing surveillance cameras in public places. Again, this tendency is evident in the available data: in 2014, [2] estimated that there were 245 million surveillance cameras, while in 2019, the estimation [3] was 770 million. These reports expect that the number will continue to increase, reaching 1 billion in the coming years. This means that there will be one surveillance camera for every 8 people in the world.

These numbers have resulted in a large amount of video footage, making it impossible to lively and accurately monitor these videos by employing human operators. It is a fact that surveillance cameras provide evidence after crimes have been committed, but they are rarely used to prevent or stop criminal activities in real-time. However, being able to do so would be highly desired, as it would dramatically reduce reaction times. In many cases, this is a key factor to prevent fatalities and/or reducing the suffering of victims.

Despite the UN being concerned about how advances in Artificial Intelligence (AI) could worsen the situation by, for example, automating the development and production of intelligent weapons [4], recent contributions of AI to this and other related fields have proven that it could be of great help to tackle it. Moreover, the irruption of Deep Learning (DL) and Machine Learning (ML) has dramatically improved the capabilities of machines to learn difficult tasks.

Nevertheless, to be able to make a real impact on public safety, AI solutions must balance performance and efficiency. Being able to rapidly alert of crimes occurring is as important, if not more, as

detecting them with the highest accuracy. However, DL is well known to require many computational resources, both for training and for inference. In addition, working with video does not make it any better, since videos contain a huge amount of high-dimensional information.

1.2. Objective

Taking all the mentioned aspects into account, the Final Degree Project (Trabajo Fin de Grado, TFG) that we are presenting in this document tries to contribute to the exposed challenges. In particular, out of the crimes happening daily in our cities, we have decided to focus on those involving violence, as they are the ones that generally cause more suffering.

To this end, we propose a novel architecture for the detection of violence in surveillance videos using DL techniques. Our main goal is to achieve the aforementioned balance: computational efficiency during training and inference, translated into reducing the number of trainable parameters, while keeping a high model accuracy. To achieve this, we will simplify as much as possible the model's input by extracting the essential information needed to detect violence: human bodies and their interaction. Hence, a real-time pose estimation model will be used. Afterward, the model will combine multiple features from the motion and interactions of human bodies in novel ways to efficiently detect violence. We will show that, with the proposed approach, state-of-the-art results are achieved with far less trainable parameters.

To reach the stated objectives, we define the following sub-objectives:

- Study the related work in the area of activity recognition and violence detection, both with classical approaches and with DL techniques.
- Explore and collect the different resources needed to train the proposed architecture. This includes the datasets and the pose estimation model.
- Design and develop an architecture based on Deep Learning that is able to detect violence in videos.
- Evaluate the different components of the architecture and compare our results with the state-of-the-art.

1.3. Document structure

In order to describe in detail our proposal, this document is organized as follows. In Chapter 1, we introduce the topic, motivate the proposed project and define the objectives. The technical concepts to understand the work are reviewed at the beginning of Chapter 2, followed a thorough review of the state-of-the-art in the violence detection task. The proposed architecture is presented in Chapter 3, with details for the main components and their connections. The performance of the architecture and its components are evaluated through Chapter 4. Finally, in Chapter 5 we present the conclusions of this project, as well as some potential improvements that could be tested in the future.

2

Fundamentals and related work

This chapter contains key ideas that are needed to understand the following chapters. Both the theoretical basis and the related work give context to how the problem of violence detection has been approached in this project.

2.1. Fundamentals

In this section, we introduce the main theoretical concepts of the proposed architecture. Due to space constraints, only the ideas that play an important role in the architecture are described thoroughly. Other parts are briefly explained and references are given to complete the description. Moreover, we recommend consulting the *Deep Learning book* [5] for a general detailed guide of ML and DL concepts.

2.1.1. Deep Learning for image processing

Convolutional Neural Networks

Two basic concepts in the image processing domain are convolutions and kernels. Considering an 2-dimensional image I as a matrix of pixels, a kernel K is a matrix of weights that is applied to the image through the convolution operation. Many neural networks libraries define the convolution operation as follows:

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n). \tag{2.1}$$

A special note should be made about this definition. Formally, the operation described in Equation (2.1) is called cross-correlation and is the same as the formal convolution but without having to flip the kernel. Thus, cross-correlation differs from the formal convolution when the kernel K is not symmetric. Figure 2.1 illustrates an example of the described convolution operation.

Although the definition and the example are provided for 2-dimensional data, images are normally

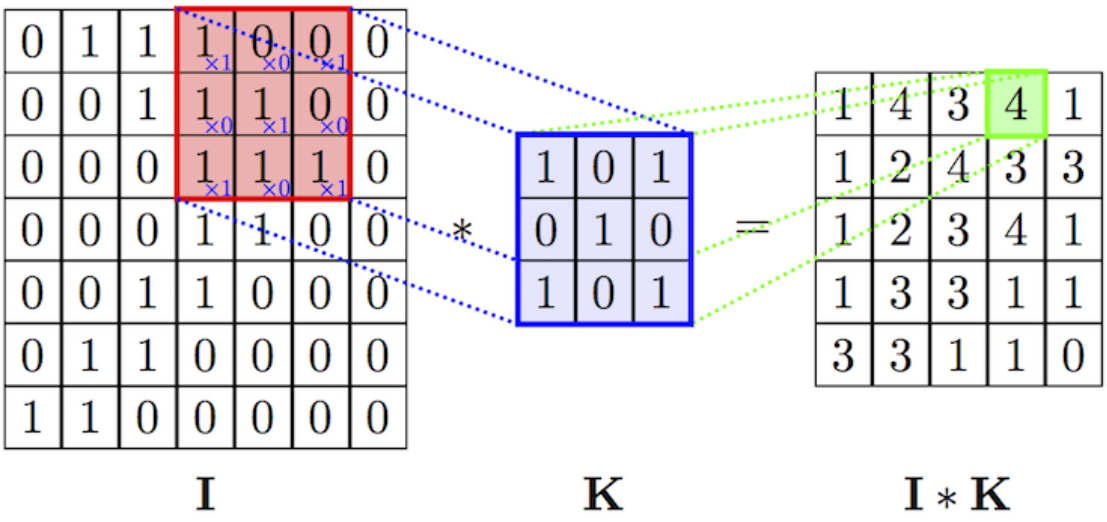


Figure 2.1: Convolution example (source: [6])

3-dimensional to encode, for example, color information. Each slice in the newly defined dimension are called channels. Later we will take advantage of the different alternatives that exist to deal with the extra dimension, so they are worth mentioning here:

1.– **Standard convolutions**: kernels are also re-defined as 3-dimensional matrices. The new convolution operation is the following:

$$(I * K)(i,j) = \sum_d \sum_m \sum_n I(i+m, j+n, d) K(m,n,d). \tag{2.2}$$

Note that the output is still a 2-dimensional matrix, as results across the last dimension are summed. This is the standard way of performing convolutions with 3-dimensional data and the default implementation in neural networks libraries.

2.– **Depthwise convolutions**: kernels are still defined as 2-dimensional matrices. However, there is at least one kernel per channel. Assuming we have exactly one per channel(that is, for each channel d, we have a 2D kernel K_d), the new convolution operation is the following:

$$(I * K)(i,j,d) = \sum_m \sum_n I(i+m, j+n, d) K_d(m,n). \tag{2.3}$$

Operating like this, information between the different channels is treated independently and is not combined in any way, which may be desired in some cases.

3.– **Depthwise Separable convolutions**: this is a combination of the previous two cases. First, a depthwise convolution is performed on each channel independently. Then, the results are combined in a pointwise weighted sum across the channels with another $1 \times 1 \times d$ kernel. Formally:

$$(I * K)(i,j,d) = \sum_m \sum_n I(i+m, j+n, d) K_d^{(1)}(m,n), \tag{2.4a}$$

$$S(i,j) = \sum_d (I * K)(i,j,d) K^{(2)}(d). \tag{2.4b}$$

One can think that the last option is just a special case of the standard convolution, where information across channels are combined in a weighted sum. Although this is the case for one filter, the benefits from this alternative come when we want to have multiple feature channels (or feature maps)

as outputs. For instance, let us consider an example where we want to extract d' feature channels from an input with d channels, considering kernels of size $M \times N$. If we use the first option, we would need d' kernels of size $M \times N \times d$. By using the last option, we would need d kernels of size $M \times N$ and d' kernels of size $1 \times 1 \times d$. Adding the numbers, the last option is clearly preferable in terms of the number of parameters: $d' \cdot M \cdot N \cdot d >> d' + M \cdot N \cdot d$. Examples in the literature show that the last option is not only more efficient but also prevents overfitting [7]. For more details about the non-standard convolutions described above, as well as novel applications of them, see [8].

Many of the classic approaches in the image processing field consisted on manually defining kernels to extract specific features [9–12]. The Deep Learning paradigm has made it possible to consider weights inside kernels as learnable parameters that can be optimized by a neural network. This leads to the definition of Convolutional Neural Networks (CNNs), that consist of a series of layers of learnable kernels (convolutional layers), typically followed by a final fully connected layer. Moreover, considering kernels and convolutions instead of usual fully connected layers enables these networks to process local spatial information. If not, we would have to flatten images to feed the network, which would lead to a huge number of parameters and losing the spatial information.

Apart from convolutional layers, CNNs include other relevant elements that are out of the scope of this section. An extensive and detailed explanation of them can be found in [5, Chapter 9]. Finally, an example of a typical CNN architecture can be found in Figure 2.2. Note that, apart from serving as an example, this is also the first ever proposed CNN.

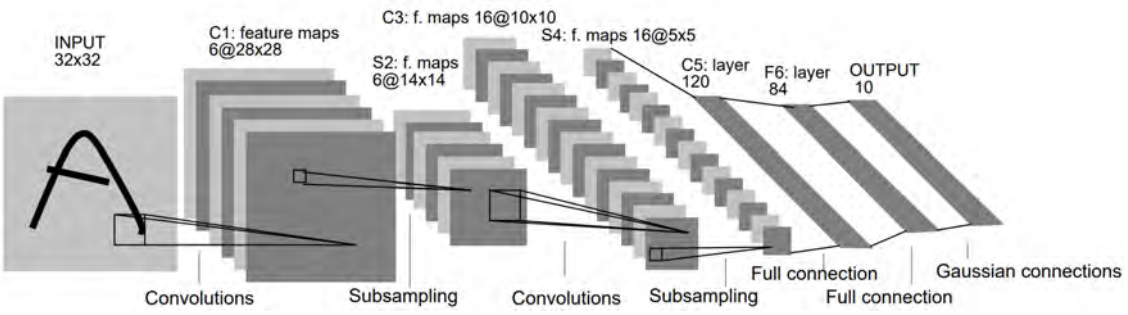


Figure 2.2: LeNet architecture (source: [13])

Global Pooling

Another concept that will play an important role in the coming chapters is *Global Pooling* [14]. It comes from the need to flatten the information contained in the feature maps to feed the final fully connected layer. One option is to directly flatten all values contained in the feature maps into a 1-dimensional vector. In some cases, this leads to a huge number of parameters, which is not desirable. Another option is to summarize the information contained in the feature maps by getting a representative value for each feature map. For instance, the maximum or the average of each feature map can be used.

Figure 2.3 illustrates the described reasoning for *Global Pooling*.

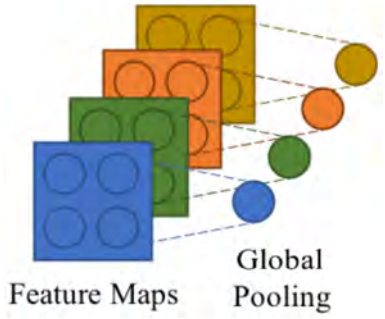


Figure 2.3: Global Pooling. Adapted from [15]

Activation functions

To model non-linearity, *activation functions* are applied to the output of each layer. There are many functions with different properties that are usually considered for this purpose, but the ones used in the next sections are the following:

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}, \tag{2.5a}$$

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}, \tag{2.5b}$$

$$\text{ReLU}(x) = \max(0, x). \tag{2.5c}$$

Note that $\text{sigmoid}(x) \in [0,1]$, $\tanh(x) \in [-1,1]$ and $\text{ReLU}(x) \in [0,\infty)$. Further details about activation functions can be found in [5, Section 6.2].

Batch Normalization

Bacth Normalization [16] was proposed to reduce the *internal covariate shift* during the training process. Updates in the parameters of a layer change the distribution of its output, which changes the distribution of the inputs to subsequent layers. These shifts in input distributions make the learning process more difficult, and Batch Normalization seeks to stabilize this behavior by normalizing the outputs before they are passed to the next layer. Therefore, outputs are forced to have a mean of 0 and a standard deviation of 1, by subtracting their mean ($\mathbb{E}[x]$) and dividing by their standard deviation ($\sqrt{\mathbb{V}[x]}$), as shown in Equation (2.6).

$$\hat{x} = \frac{x - \mathbb{E}[x]}{\sqrt{\mathbb{V}[x] + \epsilon}}. \tag{2.6}$$

During training, the required statistics are estimated from the batch of data that is being processed.

That is, if processing a batch of data $\mathcal{B} = \{x_1, \ldots, x_N\}$ of size N, the mean and variance of the batch are computed as follows:

$$\mu_{\mathcal{B}} = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{2.7a}$$

$$\sigma^2_{\mathcal{B}} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_{\mathcal{B}})^2. \tag{2.7b}$$

And so, each element of the batch is normalized as follows:

$$\hat{x_i} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}. \tag{2.8}$$

However, as mentioned in the original paper [16], normalizing may restrict what a layer can represent. For instance, normalizing the inputs of a sigmoid would constrain them to the linear regime of the non-linearity. To restore the representation power, they add learnable scaling (γ) and offset (β) factors. Taking this into account, the final output of a Batch Normalization layer is given by:

$$y_i = \gamma\hat{x_i} + \beta. \tag{2.9}$$

Having explained how the layer behaves during training, a final comment should be made about how it works during inference. We have to update how we estimate the mean and the standard deviation of the inputs, as during inference these values have to be fixed. Instead, a moving average of both statistics is computed during training and used during inference.

2.1.2. Taking time into account: ConvLSTM

Convolutional Neural Networks have proven to be excellent extractors of spatial features. However, since we are dealing with sequences of frames (videos), we have to consider an extra dimension: time. In the literature, there are many alternatives to combine spatial and temporal information, which are mentioned in the next section. One of them that is able to model spatial and temporal information at the same time is Convolutional Long Short-Term Memory (ConvLSTM) [17], so a more detailed description of ConvLSTM and its motivation is given in the following paragraphs.

As its name suggests, ConvLSTM is a variation of the well-known Long Short-Term Memory (LSTM) architecture [18]. LSTMs are a type of recurrent neural network (RNN) that were proposed to account for problems in keeping track of long-term dependencies by classic RNNs. These difficulties were first described in [19] and are mainly related to long-term gradients vanishing – i.e., tending to zero – or exploding – i.e., tending to infinity. To solve these issues, LSTM defines an accumulator of information, called *memory cell* $\mathcal{C}_t$, which is written, cleared and accessed by the input i_t, forget f_t and output o_t

gates respectively. These gates are functions of the current input $\mathcal{X}_t$ and the previous hidden state $\mathcal{H}_{t-1}$. Equations (2.10) formally describe this behavior.

$$i_t = \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + b_i), \tag{2.10a}$$

$$f_t = \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + b_f), \tag{2.10b}$$

$$o_t = \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + b_o), \tag{2.10c}$$

$$\mathcal{C}_t = f_t \odot \mathcal{C}_{t-1} + i_t \odot \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c), \tag{2.10d}$$

$$\mathcal{H}_t = o_t \odot \tanh(\mathcal{C}_t), \tag{2.10e}$$

where $*$ denotes the matrix product, $\odot$ denotes the Hadamadar product (element-wise product), σ denotes the sigmoid function and $W_{x\cdot}$, $W_{h\cdot}$ are matrices of learnable weights corresponding to the input and hidden states respectively ('$\cdot$' in the subindex refers to the corresponding gate, that is, one of i, f, o, c).

ConvLSTM updates the previous architecture, so it can process spatial information across sequences of frames by replacing matrix multiplications with convolutions and matrices of weights by kernels. Therefore, in Equations (2.10), $*$ now denotes the convolution operator and $W_{x\cdot}$, $W_{h\cdot}$ are learnable 2D convolution kernels. In Figure 2.4, we can see how the new architecture combines spatial information from the current input and the previous state, which is the one providing the temporal information.

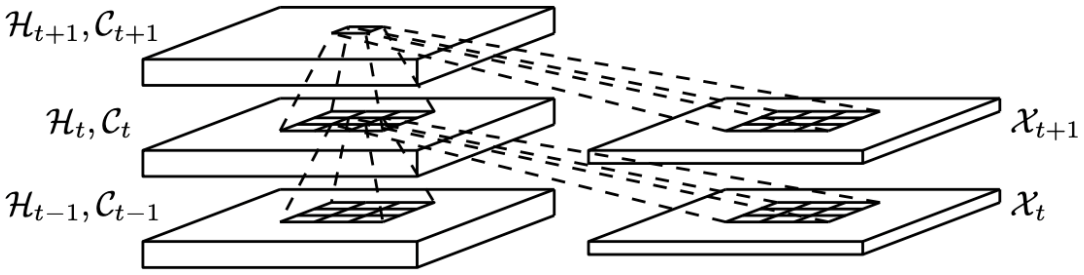


Figure 2.4: Inner structure of ConvLSTM (source: [17])

2.1.3. Estimating movement: optical flow

Optical flow is a relevant concept that has been widely used for detecting violence [7, 10–12]. Because of this, a brief description of the idea is given.

Optical flow's main goal is to estimate the movement between two frames. Each pixel of optical flow's output contains a 2-dimensional vector (dx, dy) that indicates the direction that each pixel took from one frame to the other. Figure 2.5 graphically shows this logic.

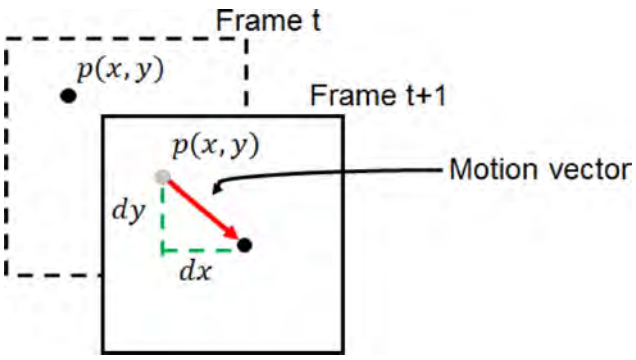


Figure 2.5: Optical flow explanation (source: [12])

Multiple methods have been proposed to estimate the optical flow. They are out of the scope of this work, but a good survey with a complete description of the classical approaches can be found in [20]. Moreover, Deep Learning has also been used recently to predict the optical flow between two frames [21, 22].

2.2. Related work

In the following lines, we describe previous approaches to detect violence in videos, as well as the available datasets that have been used.

2.2.1. State-of-the-art for violence detection

As for many Computer Vision problems, related work in the violence detection topic has been heavily influenced by the irruption of Deep Learning and Convolutional Neural Networks. Thus, we should distinguish between approaches that use classical methods and those that use Deep Learning. More importance is given to the latter, as they achieve better results. A general survey that describes works from both approaches is [23], while [24] focuses on CNN-based approaches.

Classical approaches

In the image processing field, classical approaches typically consist in extracting a numerical vector (called *descriptor*) from images in order to train standard classification models (Support Vector Machines (SVM) [5, Chapter 5], Fully Connected Networks (FCN) [5, Chapter 6], ...). Taking this into account, classical work in the field of violence detection focus their efforts on defining descriptors that gather spatio-temporal information of the videos.

Among the different proposals in the literature, the following ones give a good overview of what has been tried so far. To begin with, Nievas *et al.* [9] used two well-known descriptors in the activity

recognition field: the Spatio-temporal Interest Points (STIP) [25] and the Motion Scale Invariant Feature Transform (MoSIFT) [26]. The former is an extension of the Harris corner detector [27] which aims to detect pixels with significant intensity variation across space and time, while the latter adds motion information via optical flows to the popular SIFT [28] descriptor. Both are categorized as local invariant descriptors, which means that they are invariant to changes such as rotations, scale variations, affine transformations, ... Based on [9], MoSIFT demonstrates an overall better performance than STIP in the analyzed datasets, although it is also more computationally expensive.

On the other hand, Hassner *et al.* [10] designed a specific descriptor to represent violence information: VIolent Flows (ViF). ViF considers how optical flow changes over time, rather than the magnitude of the flows themselves. The authors claim that ViF enables real-time violence detection, which is not possible with previous descriptors. Incorporating the orientation of the flows into the ViF descriptor resulted in Oriented VIolent Flows (OViF), proposed by Gao *et al.* [11]. OViF improves ViF in non-crowded scenarios and a general performance boost is observed when combining both descriptors.

Finally, [12] proposed DiMOLIF, a descriptor that combines both local (STIP) and global (optical flow) information by analyzing the magnitude and orientation of optical flows in blocks that contain a sufficient number of STIP points. This combination and filtering of information makes DiMOLIF outperform ViF and OViF individually, but achieves similar results than when combining both.

Deep Learning approaches

Deep Learning techniques have surpassed classical approaches in many research areas. In the image processing field, Convolutional Neural Networks efficiently learn the previously handcrafted features, with greater generalization capabilities. Ding *et al.* [29] adopted these new methods and used 3D Convolutional Networks to detect violence in videos. These networks are just an extension of 2D CNNs, expanding 2D filters to the time axis and thus capturing spatio-temporal features. However, because of this expansion, 3D CNNs require many more parameters. Trying to improve on this, Li *et al.* [15] proposed an efficient 3D CNN architecture by reducing the kernel size and reusing features (each layer receives the feature maps produced by all its preceding layers).

The need to work with 3D CNNs comes from having to process not only spatial but also temporal information. An alternative to having filters managing both is to process each type of information at a time. That is, first use a 2D CNN to extract spatial features, then use a network that is able to process temporal information in these spatial features. This is what is proposed in [30], as they use a pretrained VGG16 [31] network to extract low and high level spatial features which are then fed into a LSTM.

However, a regular LSTM takes in 1-dimensional inputs, so projecting the spatial 2-dimensional features learned by the CNN may result in a loss of information. To overcome this, Sudhakaran *et al.* [32] proposed using a ConvLSTM to aggregate, across time, the spatial features produced by a pretrained CNN. Their experiments show that ConvLSTM achieves better results with fewer parameters

than a regular LSTM. Moreover, they feed difference of adjacent frames to the model instead of the raw video, as a crude but time-efficient approximation of the optical flow. Hanson *et al.* [33] extended this architecture by using bidirectional ConvLSTMs, which process the videos forward and backwards.

Lately, given the success of incorporating extra input streams of information on general activity recognition tasks [34], there have been proposals that have adopted this strategy to detect violence. For instance, [7] used two streams of 3D CNNs to process raw videos and optical flows independently, and then combined the spatio-temporal features learned in both pipelines to output the prediction. Another example is [35], which used two independent pipelines that process raw videos with their background suppressed and difference of adjacent frames, respectively. Each pipe consists of a 2D CNN and a Separable ConvLSTM (same as a ConvLSTM but with a depthwise separable convolutions). The authors explored multiple ways of combining the two pipes, with a ReLU activation of the first pipe multiplied by a Sigmoid activation of the second pipe scoring the highest in the experiments. Also, it is again noticeable the use of difference of adjacent frames as a time-efficient alternative to the optical flow.

An alternative based on very different ideas than the ones in the previous lines is described in [36]. These authors proposed to detect violence by studying the interrelationships between detected human body points. To do that, they first detect human skeletons points using a pretrained network. Afterwards, they model the interaction between points using a custom Graph Neural Network, which they called Skeleton Points Interaction Learning (SPIL) module.

2.2.2. Available datasets

Over the years, many datasets have been released for violence detection. Some of them were explicitly created for this task, while others are a subset of more general datasets for activity recognition. In the Table 2.1, one can find a comparison of all of them.

Authors	Year	Dataset	Data Scale	Length/Clip (sec)	Resolution	Annotation	Scenario
Blunsden *et al.* [37]	2010	BEHAVE	4 Videos (171 Clips)	0.24-61.92	640 × 480	Frame-Level	Acted Fights
Rota *et al.* [38]	2015	RE-DID	30 Videos	20-240	1280 × 720	Frame-Level	Natural
Demarty *et al.* [39]	2015	VSD	18 Movies (1,317 Clips)	55.3-829.4	Variable	Frame-Level	Movie
Perez *et al.* [40]	2019	CCTV-Fights	1,000 clips	5-720	Variable	Frame-Level	Natural
Nievas *et al.* [9]	2011	Hockey Fight	1,000 Clips	1.6-1.96	360 × 288	Video-Level	Hockey Games
Nievas *et al.* [9]	2011	Movies Fight	200 Clips	1.6-2	720 × 480	Video-Level	Movie
Hassner *et al.* [10]	2012	Crowd Violence	246 Clips	1.04-6.52	Variable	Video-Level	Natural
Yun *et al.* [41]	2012	SBU Kinect Interaction	264 Clips	0.67-3	640 × 480	Video-Level	Acted Fights
Sultani *et al.* [42]	2018	UCF-Crime	1,900 Clips	60-600	Variable	Video-Level	Surveillance
Cheng *et al.* [7]	2020	RWF-2000	2,000 Clips	5	Variable	Video-Level	Surveillance

Table 2.1: Comparison of available datasets. Natural refers to videos recorded by hybrid devives (e.g., mobile cameras, car-mounted cameras). Adapted from [7]

The chosen option to train and validate our architecture is RWF-2000 [7] because of the following

reasons. First and foremost, it is the dataset with the largest number of videos. Second, having videos of fixed length eases the architecture design and the training process. Moreover, as expressed in Chapter 1, our goal is to detect violence in surveillance scenarios, which are not usually monitored. The fact that videos in this dataset are recorded from surveillance cameras suits the reasoning behind our choice.

3

Proposed architecture

3.1. Architecture overview

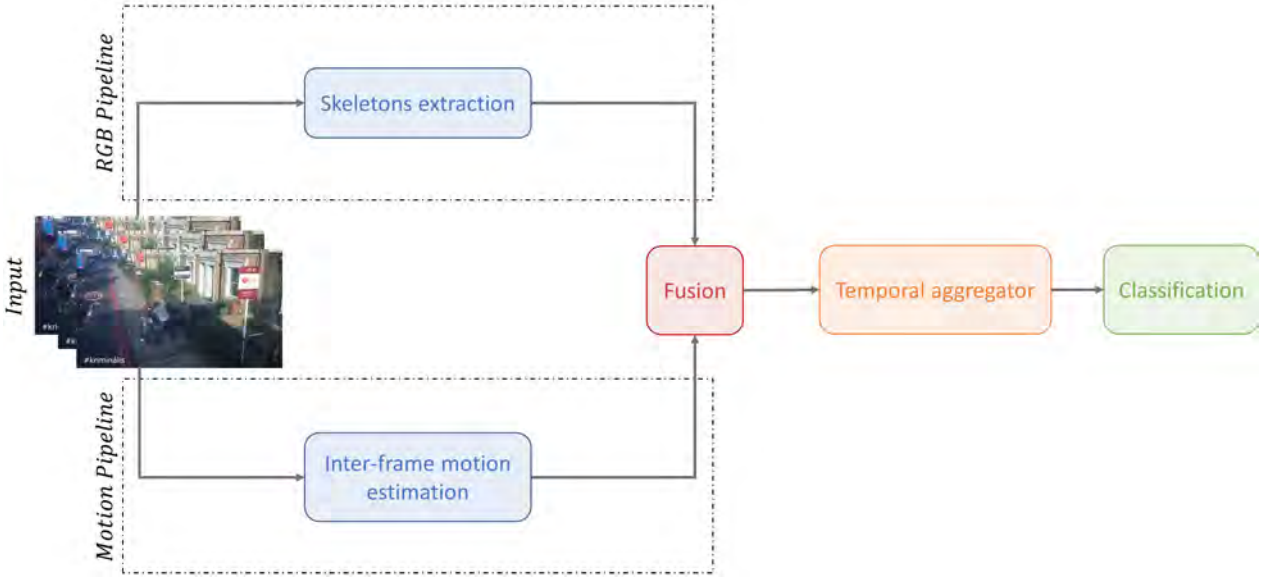


Figure 3.1: Schematic proposed architecture

A high level scheme of the proposed architecture is shown in Figure 3.1. As it can be seen, it consists of two streams of information: one in charge of extracting the skeletons of the people in the scene, and the other one in charge of estimating the movement of these people. Both have proven to be essential to detect violence, and even more powerful when combined properly. Afterwards, we compact the extracted spatio-temporal features with a temporal aggregator, followed by a final classifier that outputs the violence prediction. In the following sections, we successively detail specific components for each part, transforming the high-level scheme into a concrete proposal. Performance of the selected components and the alternatives that have been tried are evaluated in Chapter 4.

3.2. RGB pipeline

The input to this pipeline is the original sequence of RGB frames, and thus will be in charge of adding frame-level information to the architecture. Original frames may contain too much information than what is really needed to detect violence, which adds a lot of noise and makes the learning process more difficult. In this effort to keep only the essential information and reduce noise, we extract the skeletons of the people in the scene. We believe that violence involves moving certain parts of the body in particular ways, so body information, in the form of skeletons, should be enough to detect violence.

3.2.1. Human Pose Estimation: OpenPose

The area of research that tries to extract human skeletons from images is called *Human Pose Estimation*. Although there have been many proposals to solve this task, we are interested in models that are able to detect more than one person, which is called *Multi-person Pose Estimation*. Out of the available options, the selected one is OpenPose [43] because of the following reasons. First, it achieves state-of-the-art results in public benchmarks for Multi-person Pose Estimation. Second, its computing time does not depend on the number of people in the video, while other alternatives' runtime grow with the number of people. Depending on the surveillance scenario, this can be a determining factor. Third, after having tried others, OpenPose provides a very easy-to-use API, also reflected in its GitHub statistics over similar models.

A prediction example of the OpenPose model is shown in Figure 3.2. The library provides the option to render the predicted skeletons over the original video or over a black background. Since we want to remove as much noise as possible, we are very interested in the second option. Moreover, the different parts of the skeletons are colored with specific colors. This adds a valuable piece of information to the video that is going to be processed by our architecture, since it enables our model to differentiate between different parts of the body.

(a) OpenPose with background

(b) OpenPose without background

Figure 3.2: OpenPose example

After having extracted the skeletons, we use Deep Learning image processing techniques to extract spatio-temporal features from the resulting sequences of images. To our knowledge, this is the first time that such approach has been used to detect violence. The closest idea that have been found is [36], where authors studied interrelationships between human body points with a Graph Neural Network.

3.2.2. Enhancing skeletons extraction with video pre-processing

By only keeping the skeletons over a black background, we risk losing performance if the skeletons are not well detected. Authors in [44] already explored some pre-processing techniques to improve the performance of OpenPose in different settings. Their conclusions, compared to plain OpenPose, are shown in Figure 3.3. It can be seen that increasing the contrast or using histogram equalization before executing OpenPose are the options that give the best results. Based on this analysis, we decided to try both options.

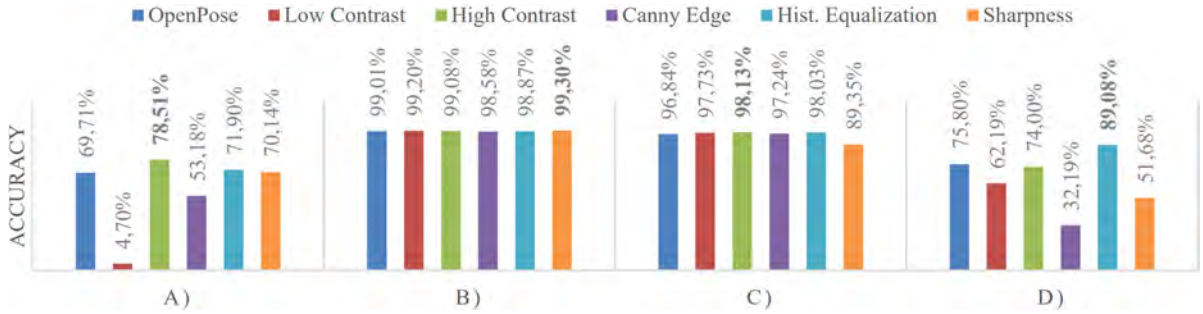


Figure 3.3: OpenPose performance for different settings: A) dark indoor scenario, B) static simple background scenario, C) dynamic background scenario, D) complex static background. (source: [44])

Contrast increase

To begin with, increasing the contrast of an image involves multiplying each pixel by a factor. However, we may not want to increase the value of each pixel by the same factor, since some areas may already be much brighter than others, or much darker than others. In the first case, we would like the increasing factor to be small, while in the second case the factor should be larger. To take this into account, we propose to use a gamma correction [45, Section 3.2]. Figure 3.4 shows the curve of the transformation. By choosing $\gamma < 1$, we achieve the desired behavior. Formally, if I is an image with pixel values in the range $[0, 255]$, the gamma correction is

$$\left(\frac{I}{255}\right)^{\gamma} \cdot 255. \tag{3.1}$$

A comparison of the output of increasing the contrast in the standard way and with the gamma correction is shown in Figure 3.5. It can be seen that gamma correction enlightens more the dark areas than the already bright ones, resulting in a better representation of the original image.

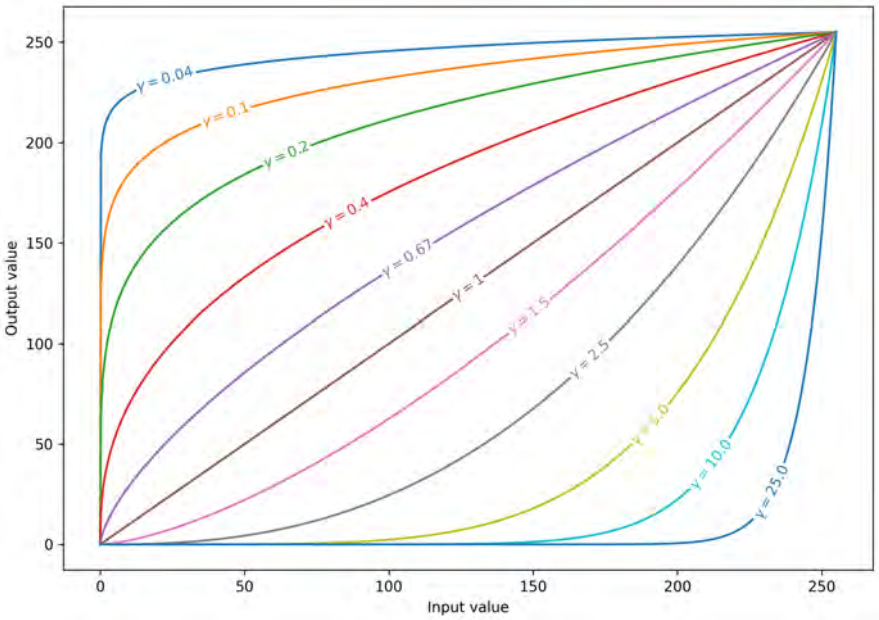


Figure 3.4: Plot for different values of gamma

(a) Original image

(b) Contrast increase (factor $= 2.0$)

(c) Gamma correction ($\gamma = 0.4$)

Figure 3.5: Comparison of contrast increase techniques

Histogram equalization

On the other hand, the other alternative to enhance the performance of OpenPose is histogram equalization [45, Section 3.3]. This technique consists of spreading the pixel values over the range $[0, 255]$, so their distribution is wider and more uniform than the one of the original image.

Figure 3.6 shows the effect of the histogram equalization on the image. The resulting image has a more uniform distribution of brightness.

(a) Original image

(b) Histogram equalization

Figure 3.6: Comparison of histogram equalization results

A few comments should be made about the mentioned pre-processing techniques. From the example outputs, we can see that both the gamma correction and the histogram equalization achieve similar outputs. However, the latter is much more computationally expensive than the former, so the gamma correction is preferred. Nevertheless, the pre-processing techniques may not be needed at all in the production environment, if the camera's resolution or the settings where it is deployed are appropriate.

3.2.3. Explicit RGB architecture

With all the ideas described in this section, we can explicitly define the RGB pipeline in the architecture scheme, as shown in Figure 3.7.

Figure 3.7: Explicit RGB architecture

3.3. Motion pipeline

Once we have extracted pose information from the original frames, it may be useful to also feed the model with movement information. Other authors [32, 33] have used movement estimators as their sole input with good overall results. There are different alternatives to estimation motion between a pair of frames, which we describe in the following lines. It has to be mentioned that, in all of them, we compute inter-frame operations between the current and the next frame in the sequence, adding future temporal information. The options are the following:

- **Frame difference**: the difference between the current and the next frame is used as the input. That is,

$$frame\_diff_t = frame_{t+1} - frame_t. \tag{3.2}$$

  Note that the result of this operation has the same number of channels as the frames. This approach was used in [32, 33].

- **Frame distance**: the distance between two frames is defined as the L^2 norm of the frame difference. That is,

$$frame\_dist_t = ||frame\_diff_t||_2 = \sqrt{\sum_{d=1}^{3} \left(frame_{t+1}^d - frame_t^d\right)^2}, \tag{3.3}$$

  where d is the channel. In this case, the result of the operation compacts the information into a single channel. In addition, all the pixels of $frame\_dist_t$ will be non-negative. [46] uses this approach.

- **Frame difference & distance**: frame difference, in comparison to frame distance, has an additional source of information: the sign. Apart from containing information about what parts of the frame are moving, by taking the sign into account one can determine from where to where each part is moving. This is because negative values indicate that there is something in the current frame that will not be there in the next one, while the positive sign indicates that there is something in the next frame that was not there in the current one. In order to get the sign information with a non-negative input, in case the model is not able to distinguish the difference between negative and positive values, we propose the following:

$$frame\_diff\_dist_t = -(frame\_diff_t)^- + ||(frame\_diff_t)^+||_2, \tag{3.4}$$

  where $(\bullet)^+$ is the positive part (i.e., $(x)^+ = \max(x, 0)$), and $(\bullet)^-$ is the negative part (i.e., $(x)^- = \max(-x, 0)$). Figure 3.8(d) shows the result of this operation and is a good explanation of the idea behind the proposed estimator. Parts that will "disappear" (negative sign) from the current frame have their original colors, while parts that will "appear" (positive sign) have constant gray colors, as a result of computing their L^2 norm across the channels. Moreover, this last part is smoothed with a Gaussian kernel, as we are interested in the area towards which the movement is going, not the exact position.

- **Optical Flow**: as mentioned in Section 2.1.3, optical flow is a way to estimate the movement that happens between two frames. Moreover, many authors have used optical flow in the action recognition field as an input to their models, and [7] specifically in violence detection. Because of these facts, we include it as a possible input to the motion pipeline. The obtained 2-dimensional results are rendered as an RGB video following the algorithm available in [47].

Figure 3.8 shows a sample output of all the proposed estimators for a specific frame. At first sight one could think that frame distance (Figure 3.8(c)) offers more information, but it has to be mentioned that in Figure 3.8(b), we are not able to visualize the full potential of frame difference because there

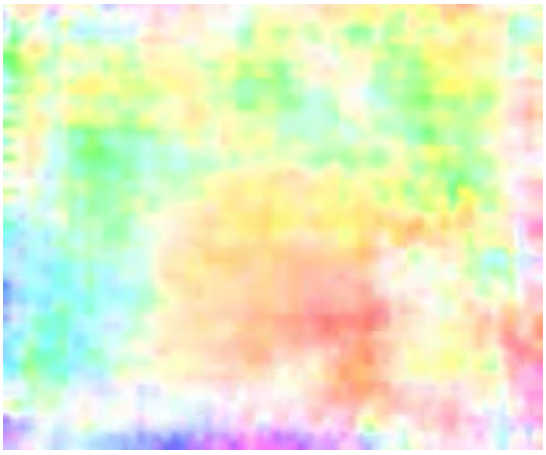
(a) Original image

(b) Frame difference

(c) Frame distance

(d) Frame difference & distance

(e) Optical flow (Farneback)



(f) Optical flow (FlowNet 2.0)

Figure 3.8: Comparison of movement estimators. The parts that are moving in the original video are the two men in the bottom right corner and the man laying down in the center

is no way to render the negative values without rescaling. The proposed combination may not be as effective as initially thought, since it seems a little noisy and too similar to frame distance. Moreover, we see that FlowNet (Figure 3.8(f)) produces much more noise than the classical Farneback's algorithm (Figure 3.8(e)). This is because of FlowNet being a model outputting a prediction, which will always contain more noise than the ground truth. Finally, after having executed this example, we could check how computationally expensive is computing optical flow.

Based on the lessons explained, frame difference appears to be the best available option for the motion pipeline and thus is chosen as the input to this pipeline, as shown in Equation (3.5). This is mostly because the sign information it provides and the computation efficiency. Nevertheless, all alternatives will be evaluated in Section 4.3.

$$input\_motion_t = frame_{t+1} - frame_t. \tag{3.5}$$

We can now explicitly define the motion pipeline, as shown in Figure 3.9.

Figure 3.9: Explicit motion architecture

3.4. Merging the two pipelines

Based on the fact that pose and movement information had good results separately, together with finding examples in the literature [7, 34, 35] that used two streams of information, we decided to incorporate both pipelines into the proposed architecture.

In order to incorporate as much movement information as possible, we decided to further preprocess the motion pipeline before merging it to the other pipeline. By applying a ConvLSTM to the frame difference before the merge, we supply the model with more information of the inter-frame relationships. Moreover, so as to match dimensions and prepare the skeletons' data for the merge, we add a convolutional layer to the first pipeline.

On the other hand, combining both sources of information would also reduce the risk of missing information when the skeletons are not well detected, as the movement pipeline could still supply information to the architecture. An example of this is shown in Figure 3.10, where skeletons of the people that are fighting (see bottom right corner of Figure 3.10(a)) are not detected but the frame difference is

able to capture this information. However, the combination has to be done in an intelligent way in order to take full advantage of both sources.

(a) Original image

(b) Skeletons detected

(c) Frame difference

Figure 3.10: Motion of merging pipelines

Authors that have previously tried using two streams explored different ways of combining them [35]. Concatenating and multiplying features are the most common options, with the latter performing better. When multiplying, the $\mathrm{sigmoid}$ activation is usually applied to one of the streams, interpreting it as a weighting factor between 0 and 1. Following this logic, we could apply the $\mathrm{sigmoid}$ function to the feature maps of the motion pipeline, so they can weight which areas of the RGB stream the model should pay more attention to. Formally,

$$F_{combined} = \mathrm{ReLU}(F_{RGB}) \odot \mathrm{sigmoid}(F_{Motion}), \tag{3.6}$$

where F_{RGB} and F_{Motion} are the corresponding feature maps.

However, multiplying has the restriction that both streams have to carry information for it to work. That is, if there is a 0 in the RGB pipeline, it will not be possible for the motion pipe to provide information, and vice versa. As we want both streams to enrich the information that is provided to the next layer,

not restricting it, we designed a novel combination technique. It is based on the following principles. First, we substitute the multiplication operation with addition, so both streams can contribute. Then, we change the $\mathrm{sigmoid}$ by the $\tanh$ function. This is because the sign of the values of the $\tanh$ (in the $[-1, 1]$ interval) can play the role of activation/deactivation when being added, that was previously played by the outputs of the $\mathrm{sigmoid}$ when multiplying. Finally, if the values of the other stream are large or with large variance, adding a -1 or a 1 will be totally irrelevant. To understand this reasoning, one can think of the impact of adding a ± 1 to values in $[0, 255]$ in comparison with values in $[-0.5, 0.5]$. Therefore, it is desired that values in the RGB pipeline are transformed to be around 0 with small variance, which can be achieved by applying a Batch Normalization layer. Note that, in Equation (2.9), we have to explicitly set $\gamma = 1$ and $\beta = 0$, since other values would modify the mean and variance of the output. The proposed combination is summarized in Equation (3.7).

$$F_{combined} = \mathrm{BatchNorm}(F_{RGB}) \oplus \tanh(F_{Motion}), \tag{3.7}$$

where $\oplus$ is the pointwise addition.

Taking the addition of the ConvLSTM to the motion pipeline and the proposed way of combining the two streams, we redefine the architecture as shown in Figure 3.11.

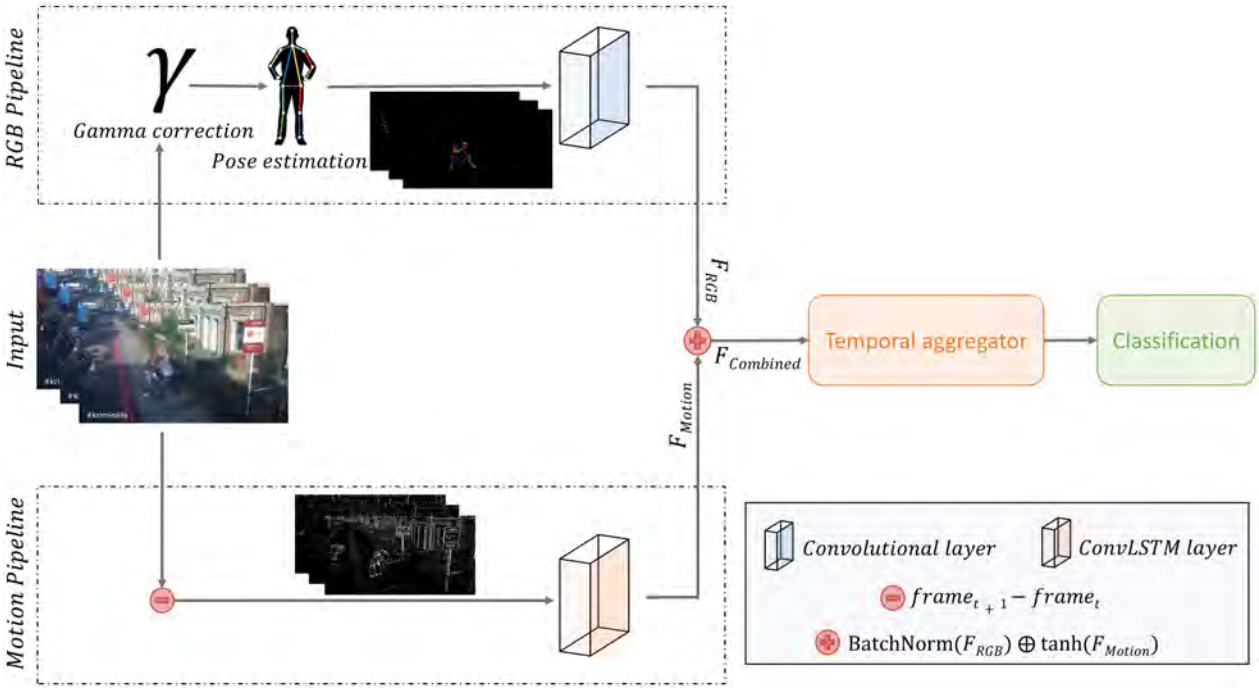


Figure 3.11: Explicit merging architecture

3.5. Aggregating spatio-temporal information and final prediction

At this point of the architecture, we have obtained a set of feature maps $F_{combined}$ for each frame in the input sequence. We would like to aggregate them into video-level features, since we have to detect if there is violence in the entire input sequence. This section is about how this aggregation is done and how it is further processed before giving the final prediction.

3.5.1. Spatio-temporal aggregator: ConvLSTM

In order to obtain the desired video-level feature map, we would like to use a layer that is able to process spatial and temporal features. As described in Section 2.1.2, the ConvLSTM is able to take both into account. Recalling what was mentioned in that section, the layer produces a state (feature maps) for each element of the input sequence. Since the last state's computation involves all the elements of the sequence, via the information accumulated in the previous state, it can be considered as the aggregated state of the entire sequence. Therefore, the ConvLSTM will output a set of feature maps, i.e. the last state of the network, as a compact video-level summary.

Keras Pull Request

Before, continuing with the definition of the architecture, a very important comment should be made about the implementation of the ConvLSTM layer when outputting just the last state of the network.

As for other layers of the architecture, we have used Keras [48]. However, while building the model architecture, an unusual great memory usage was detected when adding the ConvLSTM layer. Since Keras is an open-source library, it is possible to check the actual code. After inspecting it, a bad memory management was detected for all the recurrent layers of the library, as they were storing in memory useless copies of information.

Specifically, recalling Equations (2.10), to compute the current state $\mathcal{H}_t, \mathcal{C}_t$, you need the previous state $\mathcal{H}_{t-1}, \mathcal{C}_{t-1}$ and the current input $\mathcal{X}_t$. Sometimes it is desirable to retrieve all the computed states, so as to have the output of the layer for every single input. Many other times, we only want to get the last computed state, which will contain the spatio-temporal features of all the sequence. Keras allows the user to decide which of the two options to output. However, if we indicate to only receive the last state, Keras keeps in memory during the computation all the generated states $\mathcal{H}_t, \mathcal{C}_t$, to then only return the last $\mathcal{H}_T, \mathcal{C}_T$. This behavior is very much improvable, since only the previous state is strictly needed to compute the next one.

The previous behavior impacted all the recurrent layers of the library, but it was specially harmful

for ConvLSTM because of processing sequences of high dimensional data (frames). For instance, if a batch of 10 videos, of 150 frames each, was fed into a ConvLSTM with 15 filters, Keras was storing 22,500 states, as shown in Equation (3.8a), instead of the 150 states that are strictly required, as shown in Equation (3.8b).

$$10 \text{ videos} \cdot 150 \text{ states} \cdot 15 \text{ filters} = 22,500 \text{ states}. \tag{3.8a}$$

$$10 \text{ videos} \cdot 1 \text{ state} \cdot 15 \text{ filters} = 150 \text{ states}. \tag{3.8b}$$

An empirical comparison of the memory usage when executing the model before and after the proposed changes is shown in Table 3.1. Doing the math for the previous example (150 frames per video (FPV), 15 filters), considering frames of 100×100 with floats of 32 bits, the memory savings should be

$$100 \cdot 100 \cdot 32 \text{ bits} \cdot 22,350 \text{ states} = 7.152 \times 10^9 \text{ bits} = 0.894 \text{ GB}, \tag{3.9}$$

which is approximately what is reflected in the corresponding entry of the Table 3.1(b).

FPV	Filters	Before (GB)	After (GB)
50	5	2.2	2.1
50	10	3.8	3.6
50	15	5.5	5.2
50	20	7	6.6
50	25	8.5	8.1
50	30	10.3	9.7
50	35	11.8	11.2
50	40	13.3	12.6
50	45	14.8	14

(a) Varying number of filters of ConvLSTM

FPV	Filters	Before (GB)	After (GB)
25	15	3.1	2.9
50	15	5.5	5.2
75	15	7.8	7.4
100	15	10.6	10
125	15	12.8	12.2
150	15	15.3	14.4

(b) Varying number of FPV

Table 3.1: Empirical memory improvements with the proposed changes when training the model with batches of 10 videos. FPV denotes Frames Per Video. The memory usage of the GPU before the changes is shown in the "Before" column, and the memory usage after the changes is shown in the "After" column. Both are meassured in Gigabytes (GB).

This change was implemented and a pull request [49] was created on March 4th so other Keras users can benefit from the memory savings. The changes were approved on March 15th, and the pull request was successfully merged, as shown in Figure 3.12.

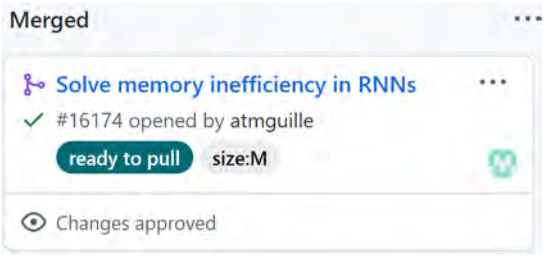


Figure 3.12: Merged Pull Request [49]

3.5.2. Refining aggregated information before prediction

Now that we have aggregated the information, we may want to post-process the obtained representation before feeding it to the final fully connected layer. It has to be taken into account that we have just collapsed a lot of information of the entire sequence into a summary set of feature maps. Therefore, refining the extracted spatio-temporal summary seems like a reasonable step before losing spatial and temporal sense in the fully connected layer.

We have two options to perform this task. First, we could consider the feature maps as independent features that should be post-processed independently. This type of independent spatial processing between feature maps (channels) can only be achieved by a Depthwise convolution, which we explained in Section 2.1.1. The other option is to consider them as a single frame with multiple channels. For this, we can apply the other two types of convolutions that we explained. Moreover, we could also use a more complex combination of successive convolutional layers to post-process the aggregated information if we consider feature maps as a single frame. Although we propose to include post-processing in the architecture, at this point it is hard to tell which option is the best without evaluating their performance. Thus, the post-processing operation is determined based on the experiments of Section 4.5.2.

Finally, we have to transform the refined feature maps in a 1-dimensional vector that could be fed to the final fully connected layer. This is done by using a *Global Average Pooling layer*, which we also described in Section 2.1.1. For the fully connected layer, we propose to use a simple architecture, trusting that the previously extracted features are sufficiently representative, without adding a significant number of computations. [35] uses $128 \rightarrow 16 \rightarrow 1$ with good results, so we will use this simple architecture.

3.5.3. Explicit final architecture

Having explained the spatio-temporal aggregator, the post-processing and the final fully connected layer, we can now describe the explicit proposed architecture as a whole. It is shown in Figure 3.13. Note that we have also added the optimum number of filters of the components that have convolutions. Determining these complexity factors, as well as evaluating all the proposed alternatives throughout this Chapter 3, is a task that corresponds to the coming Chapter 4.

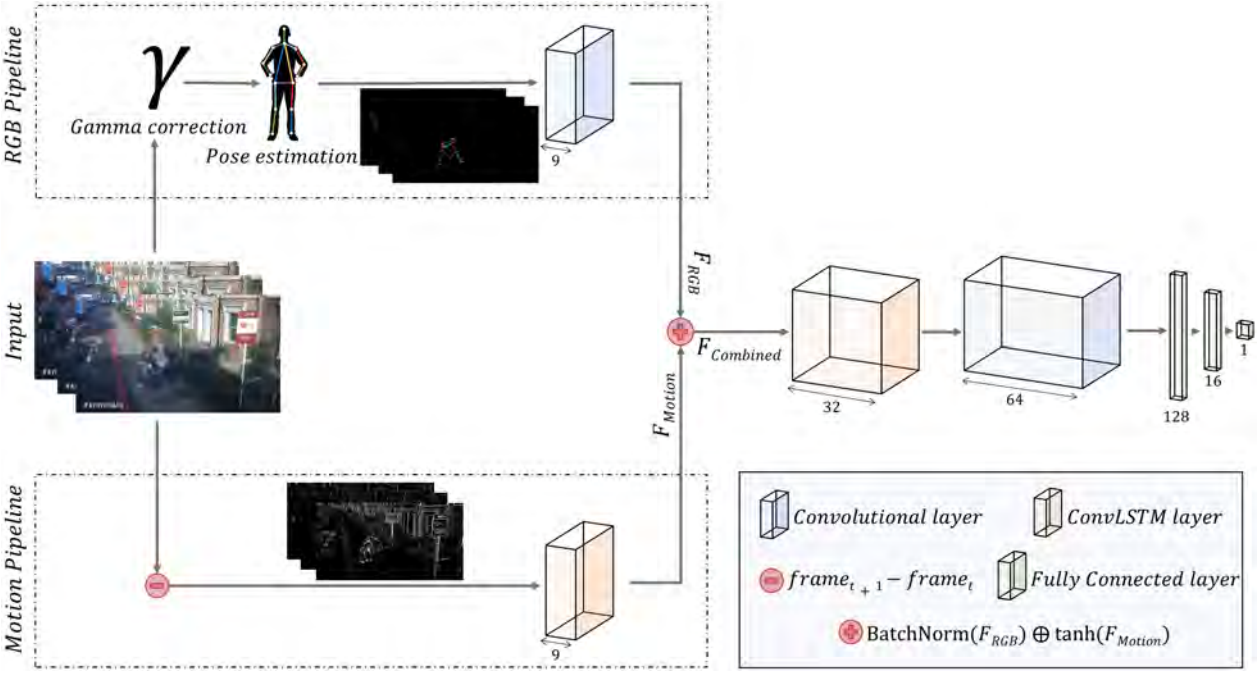


Figure 3.13: Explicit final architecture

4

Experiments

The main objective of this chapter is to evaluate the proposed architecture. To do that, we test the different alternatives for each component, as well as determining their optimum complexity. A full record of all the experiments and the code needed to execute them can be found at the following link:

http://www-vpu.eps.uam.es/publications/ViolenceDetection/

4.1. Setup for experiments

As mentioned before, the architecture is implemented using Keras. In terms of the data used for training and validating the architecture, it was said in Section 2.2.2 that the chosen dataset was RWF-2000. The dataset of 2,000 videos is completely balanced and comes with a validation split of 20 % of the videos. From each video, 50 frames equally spaced in time are extracted from the original 150 frames. That is, we reduce frames per second (FPS) from 30 to 10. Afterwards, the selected frames are resized to a dimension of 100×100. To get a sense of the type of videos our architecture will see, we have extracted a random subset of them, shown in Figure 4.1.

The network is trained with batches of 10 videos for 30 epochs. This number of epochs is chosen to be high enough to explore the full potential of the different architectures to test, but low enough to have reasonable executing time during the tests. The optimizer used is Adam [50] with a fixed learning rate of 0.001. Finally, in an effort to have reproducible results, the seed of all the random number generators is set to 0. Table 4.1(a) summarizes the training configuration. For each experiment, we show the maximum accuracy obtained in the training and the validation set over the 30 epochs, as well as the number of trainable parameters required, to give a better understanding of the complexity of each proposal. We take decisions based on performance in the validation set.

To run the experiments, we have used the hardware described in Table 4.1(b). It must be mentioned that our experiments do not require the full computing power of this machine. In fact, they perfectly run using the free resources provided by Kaggle [51] (16 GB GPU). However, the weekly time limits imposed by Kaggle restricted our ability to iterate, so we had to change to the private powerful workstation.

(a) Violent

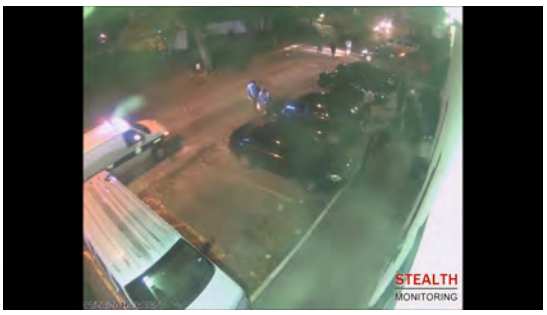
(b) Violent

(c) Violent

(d) Violent

(e) Non-violent

(f) Non-violent



(g) Non-violent

(h) Non-violent

Figure 4.1: Subset of videos from training dataset

Parameter	Value
FPS	10
Frame size	100×100
Batch size	10
Epochs	30
Optimizer	Adam
Learning rate	0.001
Seed	0

(a) Training configuration

Component	Specifications
CPU	AMD Ryzen Threadripper PRO 3975WX 32-Cores
GPU	NVIDIA A100 80 GB
RAM	256 GB
OS	Ubuntu 20.04 LTS

(b) Hardware specifications

Table 4.1: Training settings

4.2. RGB pipeline analysis

To begin with, we will analyze the contribution of the RGB pipeline. To do that, we remove the motion pipeline and evaluate performance of different options for the RGB pipeline. The architecture used in this section is shown in Figure 4.2. Note that the convolutional layer that precedes the merging is also removed, as its objective was to adapt the input to the merging layer.

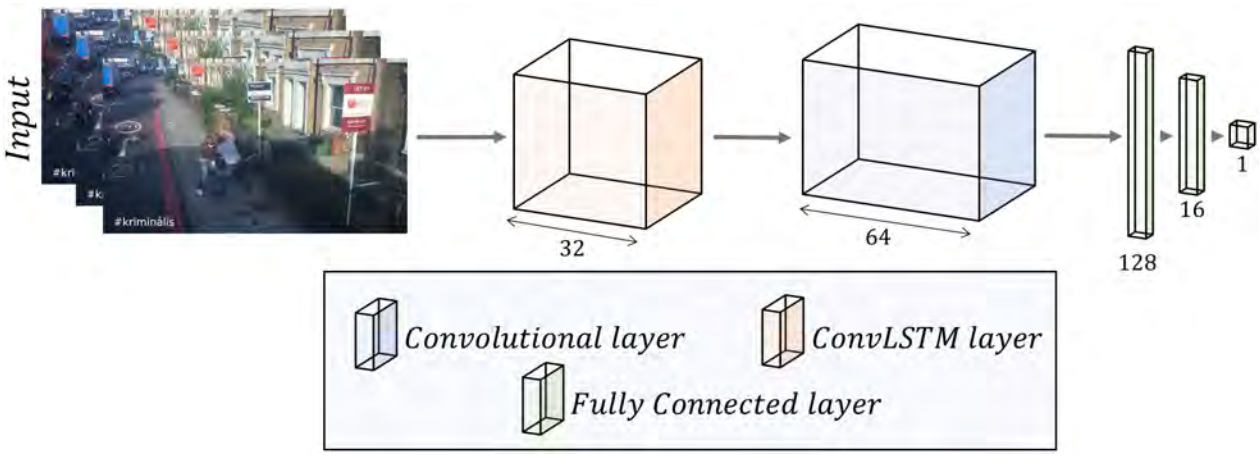


Figure 4.2: RGB architecture for the tests

4.2.1. Results for different inputs

Table 4.2 shows the performance of the architecture of Figure 4.2 with different inputs. First, we see that the architecture performs relatively well when working with the raw videos. So, even with a lot of extra noise, the model is able to learn something. If we add the pose information via OpenPose, the performance increases even with the background noise. However, in both cases, there are significant variations between the performance in the train and in the validation set, due to the noise. Finally, removing all the noise and only keeping the essential information, the skeletons, there is a significant

boost in the model accuracy, both in the train and in the validation set. This strengthens our hypothesis that analyzing human bodies and their interactions should be enough to detect violence, and that the model benefits from the reduction of the amount of noise.

Model	Train accuracy	Validation accuracy	Trainable parameters
Original videos	75.06 %	78.50 %	51,489
OpenPose with background	73.50 %	80.00 %	51,489
OpenPose without background	87.88 %	**84.75 %**	51,489

Table 4.2: Results for different inputs

4.2.2. Results for video pre-processing

As we mentioned when describing the proposed architecture in Section 3.2, we risk losing information if skeletons are not well detected. So as to maximize the performance of the OpenPose model in our videos, we test two pre-processing operations: gamma correction and histogram equalization. In particular, gamma correction is done with $\gamma = 0.67$ and histogram equalization is computed with the Contrast Limited Adaptive Histogram Equalization (CLAHE) implemented by OpenCV [52].

The results are shown in Table 4.3. While histogram equalization does not provide any improvements over not using it, gamma correction results in a great performance boost.

Model	Train accuracy	Validation accuracy	Trainable parameters
OpenPose	87.88 %	84.75 %	51,489
Gamma correction + OpenPose	88.13 %	**85.50 %**	51,489
Histogram equalization + OpenPose	86.19 %	84.75 %	51,489

Table 4.3: Results for video pre-processing

4.2.3. Results for skeletons' post-processing

Once skeletons have been extracted and rendered over the black background, we may want to apply some post-processing to the resulting images before they are fed to the next layer. A recurrent technique used in the literature [30] is using a pretrained CNN to extract spatial features before taking temporal information into account. We tried this approach by applying an EfficientNet [53], pretrained in ImageNet [54], to each frame before feeding them to the ConvLSTM. As shown in Table 4.4, the accuracy is much worse than when not applying it, apart from adding a lot of extra trainable parameters. We believe that this is because these models are pretrained with real-world images, and although the rendered skeletons resemble real human bodies, the pretrained network is not able to add new information.

Model	Train accuracy	Validation accuracy	Trainable parameters
Gamma correction + OpenPose	88.13 %	**85.50 %**	51,489
Gamma correction + OpenPose + EfficientNet	80.56 %	67.75 %	1,517,857

Table 4.4: Results for skeletons' post-processing

4.3. Motion pipeline analysis

After analyzing different alternatives for the RGB pipeline, we now move on to the motion pipeline. Again, to understand its potential, we remove the other pipeline from the architecture. Moreover, the ConvLSTM that precedes the merging is also removed. This is because we want the architecture to be as similar as possible to the one used for the RGB pipeline, in order to be able to compare the results. In fact, the testing architecture is the same as the shown in Figure 4.2, with the addition of the motion estimators right after the input.

4.3.1. Results for different movement estimators

Model	Train accuracy	Validation accuracy	Trainable parameters
Frame difference	87.75 %	**83.75 %**	51,489
Frame distance	84.38 %	81.75 %	49,185
Frame difference & distance	80.06 %	81.25 %	51,489
Optical Flow (farneback)	84.87 %	80.50 %	51,495
Optical Flow (flownet)	80.19 %	79.25 %	51,495

Table 4.5: Results for different movement estimators

Mainly, what is going to be evaluated is the potential of having movement information as the sole input of the architecture. To do so, the movement estimators that were described in Section 3.3 have been tried. Moreover, we have done experiments with two ways of computing the optical flow: a classical one (Farneback's algorithm [55], implemented in OpenCV) and a Deep Learning one (FlowNet 2.0 [22]). Table 4.5 shows the architecture performance for each estimator. A few lessons are learned from these results. First and foremost, movement information is much better than raw videos, but not as good as pose information. Out of the different alternatives, frame difference is the most effective. We conclude that the model is able to extract the sign information described in Section 3.3, as frame difference performs better than frame distance. Moreover, our proposal to make this sign information clearer for the model appears to not work as expected, probably adding too much noise. Furthermore, the optical flow is not as effective as the other simpler estimators. These are good news, as both methods that compute the optical flow are very computationally expensive. Finally, the Deep Learning approach, which is less expensive than the classical Farneback algorithm, appears not to be as effective as the classical one (as we could already see in Figure 3.8(f)).

4.4. Merging the two pipelines analysis

Once the potential of each pipeline has been evaluated separately, it is time to study the influence of the components that merge these pipelines. To do so, we simplify the architecture to its minimal form and start adding components to it. Thus, we start with the architecture shown in Figure 4.3. Note the absence of the ConvLSTM layer of the motion pipeline and the post-processing layer. The temporal aggregator is kept as it is an essential component.

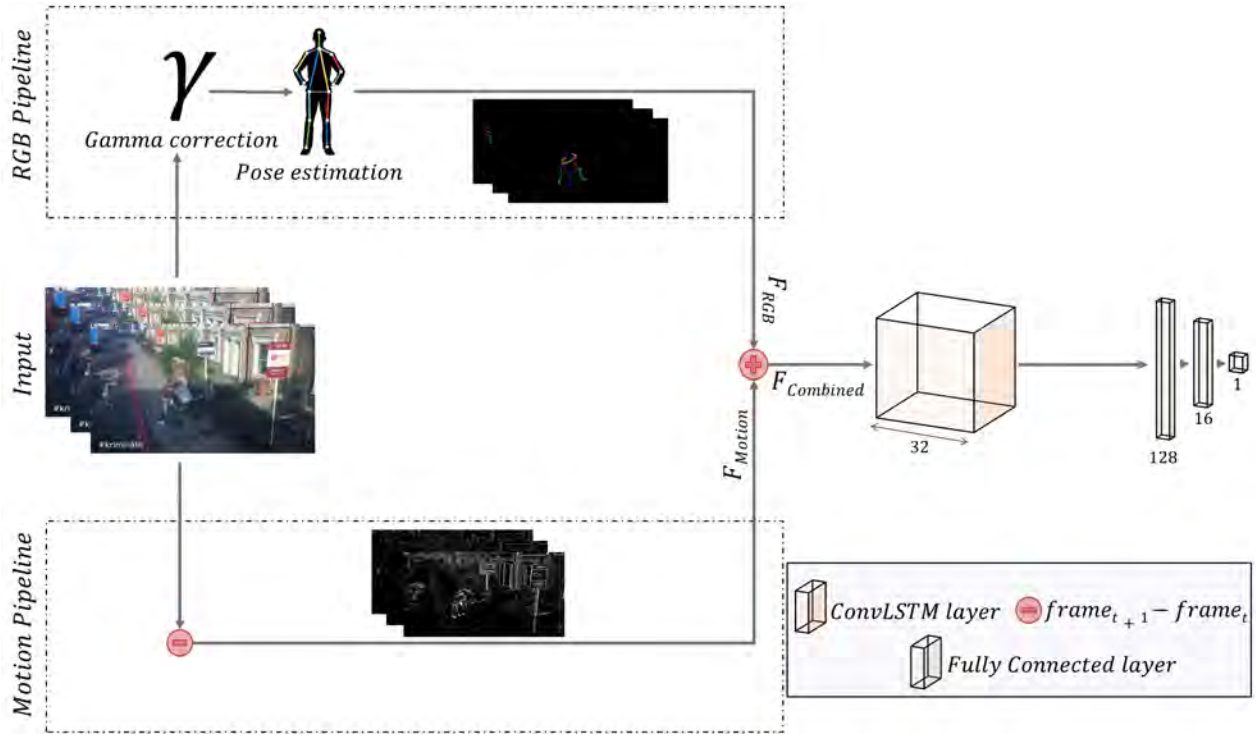


Figure 4.3: Minimal architecture to start testing

4.4.1. Results for different ways of weighting

As mentioned in Section 3.4, we propose a novel technique that aims to combine the two pipelines in a more effective way than other authors have done in the literature. Both alternatives were formally described in Equations (3.6) and (3.7). To test the combination, we use both the frame difference and the optical flow as the input of the motion pipeline. This is done because they are two radically different ways of estimating movement, and it may be the case that one performs well individually and the other performs well as a complement of the skeletons. Figure 4.3 shows the architecture used for these tests.

Table 4.6 shows that our proposed combination method outperforms by a great margin the previous combination alternatives. Furthermore, both the frame difference and the optical flow perform similarly, although we prefer the former based on its lower computation cost.

Model	Train accuracy	Validation accuracy	Trainable parameters
Frame difference + Multiply(F_{RGB}, F_{Motion})	85.00 %	85.50 %	46,921
Optical Flow + Multiply(F_{RGB}, F_{Motion})	85.87 %	85.25 %	46,927
Frame difference + Add(F_{RGB}, F_{Motion})	88.94 %	**87.00 %**	46,921
Optical Flow + Add(F_{RGB}, F_{Motion})	88.19 %	**87.00 %**	46,927

Table 4.6: Results for different ways of weighting

4.4.2. Influence of temporal information before merging

In Section 3.4 it was also mentioned that we decided to extract further temporal information by using a ConvLSTM in the motion pipe before the merge. In this section, we are going to study the influence of this addition and the optimal complexity level of the new component, which we will call preConvLSTM. The architecture to be tested in shown in Figure 4.4. Note that, as commented in Section 3.4, we have to add a convolutional layer to the RGB pipeline to match the new dimension of feature maps.

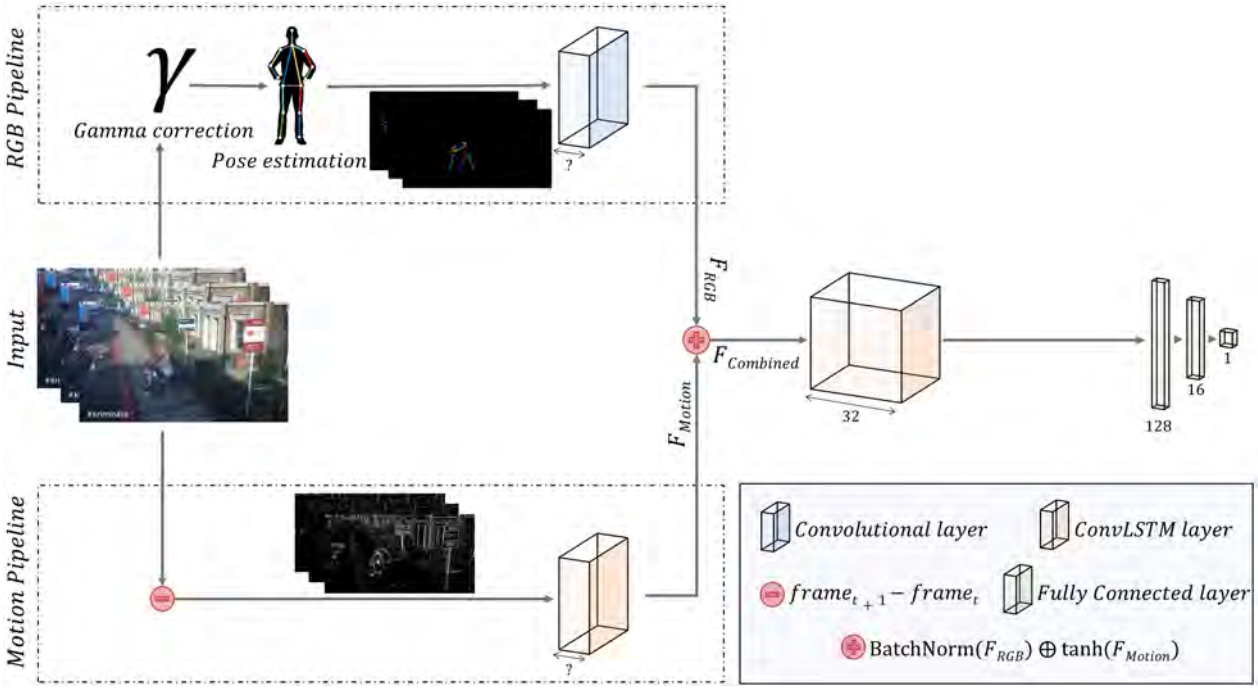


Figure 4.4: Architecture with preConvLSTM

Table 4.7 shows that the performance is positively affected by the presence of the preConvLSTM. Additionally, selecting the right amount of filters of this layer appears to be important to avoid overfitting. They were chosen as a multiple of the 3 input channels.

Model	Train accuracy	Validation accuracy	Trainable parameters
NO preConvLSTM	88.94 %	87.00 %	46,921
preConvLSTM 3 filters	88.69 %	87.50 %	47,503
preConvLSTM 9 filters	90.94 %	**88.75 %**	57,847
preConvLSTM 15 filters	92.94 %	87.75 %	70,783

Table 4.7: Influence of temporal information before merging

4.5. Aggregating spatio-temporal information analysis

Once both pipelines have been merged, it is time to aggregate the information into video-level features and output a prediction. This section analyzes the influence of the different components that are used in this process.

4.5.1. Results for different aggregator complexity

The first step is to aggregate the merged information. We have already explained that a ConvLSTM layer is the best choice, as it processes spatio-temporal information. We can adapt the complexity of this aggregator by modifying the number of filters that the ConvLSTM layer has. Therefore, the architecture being tested is the same as the one of the previous section (Figure 4.4), but with 9 filters for the preConvLSTM and varying the number of filters of the ConvLSTM. This is what is evaluated in Table 4.8. Although 32 filters is the best choice, it is also noticeable the high accuracy achieved by the simplest aggregator (16 filters). With roughly 20,000 trainable parameters, the architecture is able to perform very well.

Model	Train accuracy	Validation accuracy	Trainable parameters
ConvLSTM 16 filters	91.62 %	88.00 %	22,903
ConvLSTM 32 filters	90.94 %	**88.75 %**	57,847
ConvLSTM 64 filters	94.25 %	87.75 %	183,031

Table 4.8: Results for different aggregator complexity

4.5.2. Refining aggregated information before prediction

As noted in Section 4.5.2, there are two ways to post-process the output of the aggregator: by considering the feature maps as a single frame with multiple channels, or as independent features. In the following lines, we will explore alternatives for both options. Figure 4.5 shows the architecture that is being tested. Visually, the component under study is the one marked with a question mark.

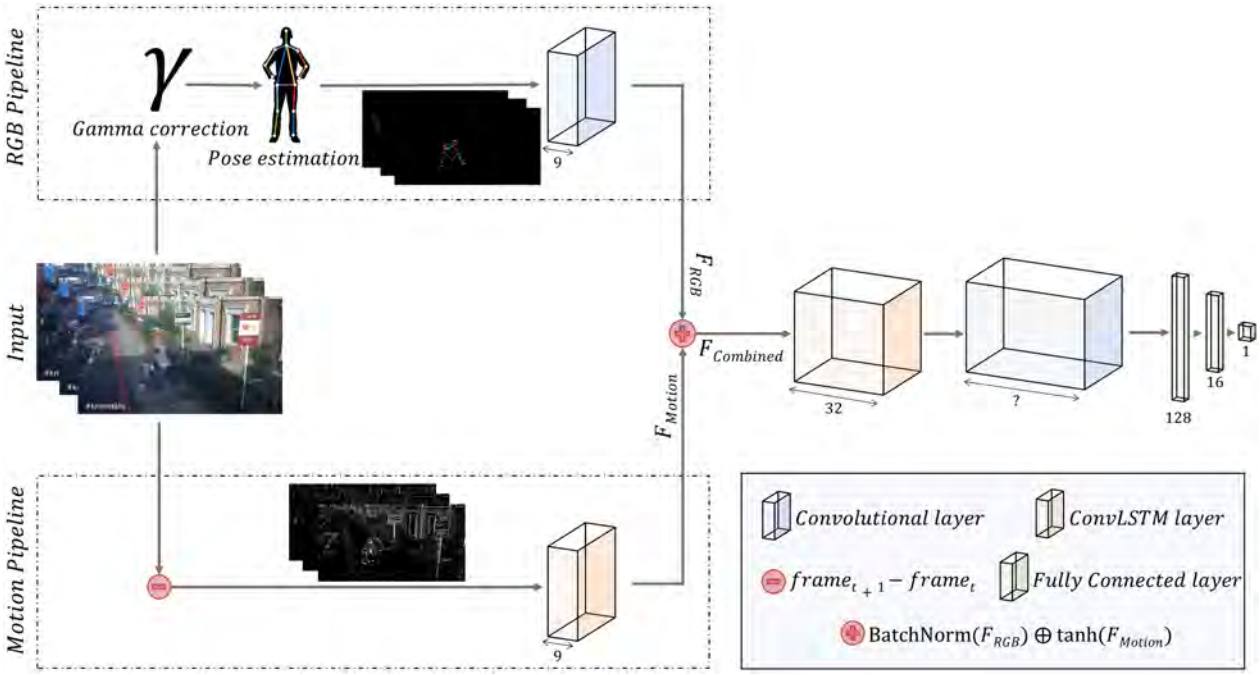


Figure 4.5: Architecture with post-processing

Spatially processing with a CNN

If we consider the feature maps as a single frame with multiple channels, we can use a CNN to process the information spatially. In this line, we could try to define our own CNN or use a pretrained one. Both alternatives are evaluated in Table 4.9. Apart from adding a huge amount of trainable parameters, none of the options performs well.

Model	Train accuracy	Validation accuracy	Trainable parameters
Custom CNN	99.44 %	85.00 %	162,031
EfficientNet	86.69 %	86.50 %	709,978

Table 4.9: Results for different CNNs

Processing information across channels

Instead of a Deep CNN, we could just use a single convolutional layer to process the information across channels. We explore the different types of convolutions defined in Section 2.1.1. The output of all the tested alternatives have been set to 64, as we wanted to post-process and augment the information. Further analysis of the optimal number is performed later.

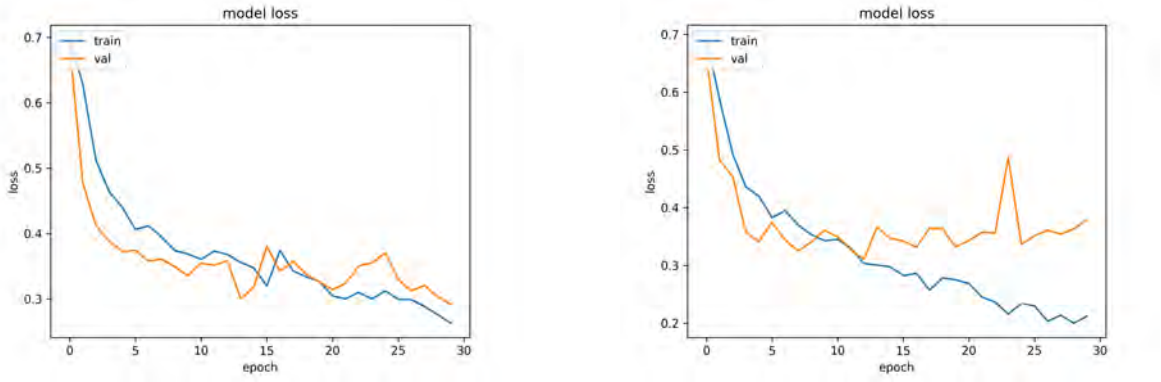
Results for the alternatives are shown in Table 4.10. First, the Depthwise convolution appears to be the one that works best. Since it processes channels independently, we can conclude that feature maps contain independent information that should not be mixed. On the other hand, we should also notice the number of trainable parameters required for each type of convolution. As mentioned in Section

2.1.1, Separable convolutions need less parameters than the Standard ones, being a less expensive alternative that also prevents overfitting. This is confirmed by results in Table 4.10. Finally, performance is very similar between Depthwise and Separable convolutions. This is reasonable, as the latter just combines information pointwise across the channels that are outputted by the former. Thus, it could be able to learn that the best combination is just not combining information across channels.

Model	Train accuracy	Validation accuracy	Trainable parameters
Standard Convolution (64 filters)	94.19 %	88.75 %	80,439
Depthwise Convolution (2 filters per channel)	89.88 %	**89.50 %**	62,583
Separable Convolution (64 filters)	92.81 %	89.25 %	64,343

Table 4.10: Results for different final convolutions

To convince ourselves that the Depthwise option is the best option, since both results are very tight in validation, we present in Figure 4.6 the evolution during training of both loss functions. We see in Figure 4.6(b) that there are clear signs of overfitting, while in Figure 4.6(a) we see that the model is able to generalize very well.



(a) Loss of Depthwise convolution

(b) Loss of Separable convolution

Figure 4.6: Loss functions of convolution alternatives

Once that we have selected the Depthwise convolution as the post-processing layer, we can analyze its performance for different levels of complexity. Table 4.11 shows that there is boost in performance when including this post-processing layer. Particularly, for the optimal number of filters, the model is able to generalize very well since the overfitting is greatly reduced. Note that with just 1 filter per channel, that is, not augmenting the information, the result is equivalent to not doing any post-processing. Moreover, not many filters per channel should be added, as it starts to lose its effectiveness.

Model	Train accuracy	Validation accuracy	Trainable parameters
No DepthConv	90.94 %	88.75 %	57,847
DepthConv 1 filter per channel	92.31 %	88.75 %	58,167
DepthConv 2 filters per channel	89.88 %	**89.50 %**	62,583
DepthConv 3 filters per channel	92.00 %	87.75 %	66,999

Table 4.11: Results for different complexity of DepthConv

4.5.3. Results for different final network complexity

After having post-processed the aggregated features, we use a Global Average Pooling layer to transform the set of feature maps into a 1-dimensional vector. This vector is fed to a fully connected layer that outputs the final prediction. In Table 4.12, we analyze the performance of the architecture with different levels of complexity for this final layer. We recall that our selected option was based in [35]. An alternative with the same number of layers but fewer neurons and another one with an extra layer with more neurons have been tried. It can be observed that a significant simple layer is able to achieve a high accuracy, demonstrating the power of the architecture that precedes the final layer.

Fully connected architecture	Train accuracy	Validation accuracy	Trainable parameters
$32 \rightarrow 16 \rightarrow 1$	92.81 %	87.75 %	54,807
$128 \rightarrow 16 \rightarrow 1$	89.88 %	**89.50 %**	62,583
$1024 \rightarrow 128 \rightarrow 16 \rightarrow 1$	93.69 %	86.50 %	252,023

Table 4.12: Results for different final network complexity

4.6. Evolution of performance summary

In order to get a final view of the evolution of the performance when adding the different components to the minimal architecture described in Figure 4.3, we summarize the obtained results throughout these sections in Table 4.13.

Proposed combination	preConvLSTM	DepthConv	Validation accuracy	Trainable parameters
✓			87.00 %	46,921
✓	✓		88.75 %	57,847
✓	✓	✓	**89.50 %**	62,583

Table 4.13: Performance evolution summary

4.7. Comparison with the state-of-the-art

Having defined the final architecture and its performance, we can now compare it with the state-of-the-art. To do so, our best model is first trained during 50 epochs, so as to achieve the maximum accuracy. After this training, our proposal achieves a maximum accuracy of 90.25 % in the validation dataset. The evolution of the accuracy during the training is shown in Figure 4.7.

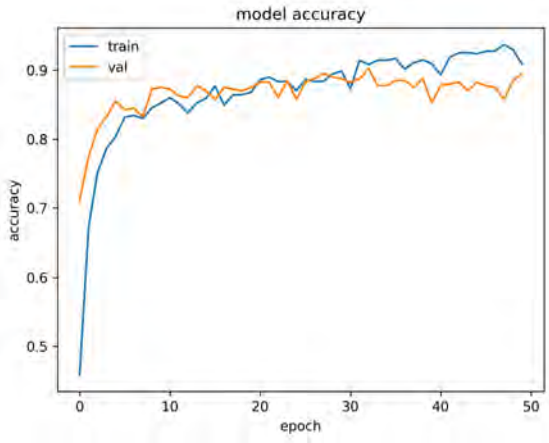


Figure 4.7: Final model accuracy evolution

Once we have our best model trained, we can compare its performance with the state-of-the-art's approaches that have provided results in the dataset that we have used. As seen in Table 4.14, our model improves all the proposals by a large margin in terms of the trainable parameters. Moreover, our accuracy is not hurt by our great reduction in the number of trainable parameters. In fact, we also surpass the state-of-the-art's accuracy of 89.75 % in the validation dataset. A special note should be made about the results of *Spatio-Temporal Modeling* [46]. Although the accuracy in their published paper was 92.00 %, there have been reports [56] that state that their real accuracy is in fact lower: 88.50 %.

Model	Year	Validation accuracy	Trainable parameters
Flow Gated Net [7]	2020	87.30 %	272,690
SPIL [36]	2020	89.30 %	–
SepConvLSTM [35]	2021	89.75 %	333,057
Spatio-Temporal Modeling [46]	2021	92.00 %* (88.50 %)	1,300,000
Ours	2022	**90.25 %**	**62,583**

Table 4.14: Comparison with state-of-the-art. '–' indicates that the number of parameters was not published.'*' means that score in the published paper appears to be lower [56]

5

Conclusions and future work

5.1. Conclusion

As we began saying in the first lines of this document, automatically detecting violence in surveillance videos is a critical task with huge potential benefits to society. Throughout this document, we have presented an architecture that is able to efficiently and accurately perform this challenging endeavor.

The efficacy of the proposed architecture is a proven fact, since we have reached state-of-the-art results with much fewer trainable parameters than other alternatives. To achieve these results, we first extracted the most essential information from the videos: people skeletons, via a pose estimation model. Then, we decided to add another important source of information to the model: the motion of the people in the scene. Each pipeline appeared to be able to detect violence independently, but were even more powerful when combined. However, this combination has to be done with care, so as to get the most out of each pipeline. Hence, we have proposed a novel combination technique that works significantly better than previous alternatives.

In personal terms, this project has been an outstanding learning experience. To begin with, we have discovered a new method that is able to process spatial and temporal information: ConvLSTMs. There are very well known techniques that are used to extract one or the other independently, but ConvLSTM intelligently mixes approaches from both sides to build a powerful trainable tool. Furthermore, we have realized that the optimality of external implementations is sometimes taken for granted, and that this is not always the case. However, these cases a great opportunity to learn and contribute.

On the other hand, we have learned what it takes to do a scientific research project. From the ability to deeply understand what others have previously tried, to the intuition required to supply the model with valuable information and the creativity to propose novel techniques; very valuable skills have been acquired. Moreover, once the proposed architecture has been implemented and experiments have been done to validate it, we have understood how important it is to clearly communicate the proposals and results. One of the foundations of the scientific community is the sharing of knowledge. For this to be possible, it must be transmitted clearly and concisely, something that we have tried to do throughout this document.

5.2. Future work

Although the objectives initially set for this project have been successfully met, there are still some potential improvements that could be tested in the future.

One of the things that we have recurrently mentioned along the document is that the performance of the architecture is very sensitive to the correct extraction of the skeletons. We have explored alternatives to enhance the performance of the extractor, as well as to supply the architecture with information if they are not correctly detected. Nevertheless, given the importance of this component, more efforts could be done to further improve its performance. First, although we consider OpenPose to be the best available option, we could explore other pose estimation. AlphaPose [57] should be the next alternative, as it achieves better performance than OpenPose in some benchmarks. Moreover, independently of the pose estimation method used, we detected that the extracted skeletons are not stable between frames. This is because the extractor is executed in each frame independently, so small errors in the locations of predictions derive in instability. This may hurt the architecture when trying to extract the interrelationships between the skeletons. Therefore, to mitigate the possible effects of this instability, smoothing the transitions of skeletons between frames could be tried.

On the other hand, we have proven that it is possible to efficiently detect violence with a limited number of trainable parameters. However, there are some alternatives that could reduce even more the number of parameters. For instance, as successfully tried in [35], we could replace the convolutional layers of the ConvLSTM layer with separable convolutions to implement a SepConvLSTM. Another way of increasing the performance without increasing the number of parameters is using more computationally cheap pre-processing. For example, the gamma correction has appeared to work well enhancing the pose estimation, so it may also be tried in the motion pipeline. In contrast, we could also try to push the accuracy with a more complex architecture, in case the model needs to be more precise than efficient.

In the near future, we plan to also validate the performance of our trained model in other public datasets. The objective of the project is to detect violence in surveillance videos, where it could help the most, but analyzing failure cases in other types of violent videos could help us to understand how our current architecture could be improved.

Finally, for this project to have the impact it was built for, it has to be able to be deployed in real settings. Results demonstrate that we have built a strong and efficient architecture for the violence detection task, but the effort will not be worth it until it detects violence and prevents crimes in real life. Therefore, our first priority is to build a working efficient system that uses our trained model to detect violence in live video streams. To do so, we plan to use OpenVINO [58], an open-source toolkit developed by Intel that optimizes trained models for fast inference in CPU or GPU.

Bibliography

[1] United Nations Office on Drugs and Crime, "Global study on homicide," 2019.

[2] N. Jenkins, "Video surveillance: new installed base methodology yields revealing results," white paper, IHS Markit - Video Surveillance Group, 2016.

[3] O. Philippou, "Video surveillance installed base report," white paper, IHS Markit - Video Surveillance Group, 12 2019.

[4] United Nations, "A new era of conflict and violence." https://www.un.org/en/un75/new-era-conflict-and-violence, 2019. Accessed: 2022-05-06.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org, Accessed: 2022-05-01.

[6] I. S. Mohamed, *Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques*. PhD thesis, 09 2017.

[7] M. Cheng, K. Cai, and M. Li, "Rwf-2000: An open large scale video database for violence detection," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 4183–4190, IEEE, 2021.

[8] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, 2017.

[9] E. Bermejo Nievas, O. Deniz Suarez, G. Bueno García, and R. Sukthankar, "Violence detection in video using computer vision techniques," in *Computer Analysis of Images and Patterns* (P. Real, D. Diaz-Pernil, H. Molina-Abril, A. Berciano, and W. Kropatsch, eds.), (Berlin, Heidelberg), pp. 332–339, Springer Berlin Heidelberg, 2011.

[10] 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*, pp. 1–6, 2012.

[11] Y. Gao, H. Liu, X. Sun, C. Wang, and Y. Liu, "Violence detection using oriented violent flows," *Image and Vision Computing*, vol. 48-49, pp. 37–41, 2016.

[12] A. Ben Mabrouk and E. Zagrouba, "Spatio-temporal feature using optical flow based distribution for violence detection," *Pattern Recognition Letters*, vol. 92, pp. 62–67, 2017.

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[14] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2014.

[15] J. Li, X. Jiang, T. Sun, and K. Xu, "Efficient violence detection using 3d convolutional neural networks," in *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–8, 2019.

[16] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 448–456, PMLR, 07–09 Jul 2015.

[17] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, “Convolutional lstm network: A machine learning approach for precipitation nowcasting,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’15, (Cambridge, MA, USA), p. 802–810, MIT Press, 2015.

[18] F. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with lstm,” *Neural computation*, vol. 12, pp. 2451–71, 10 2000.

[19] J. F. Kolen and S. C. Kremer, *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, pp. 237–243. 2001.

[20] J. Barron, D. Fleet, and S. Beauchemin, “Performance of optical flow techniques,” *International Journal of Computer Vision*, vol. 12, pp. 43–77, 02 1994.

[21] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox, “Flownet: Learning optical flow with convolutional networks,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2758–2766, 2015.

[22] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, “Flownet 2.0: Evolution of optical flow estimation with deep networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1647–1655, 2017.

[23] B. Omarov, S. Narynov, Z. Zhumanov, A. Gumar, and M. Khassanova, “State-of-the-art violence detection techniques in video surveillance security systems: a systematic review,” *PeerJ Computer Science*, vol. 8, p. e920, 04 2022.

[24] A. Jain and D. K. Vishwakarma, “State-of-the-arts violence detection using convnets,” in *2020 International Conference on Communication and Signal Processing (ICCSP)*, pp. 0813–0817, 2020.

[25] I. Laptev, “On space-time interest points,” *International Journal of Computer Vision*, vol. 64, pp. 107–123, Sep 2005.

[26] M.-Y. Chen and A. Hauptmann, “MoSIFT: Recognizing Human Actions in Surveillance Videos,” 8 1995.

[27] C. G. Harris and M. J. Stephens, “A combined corner and edge detector,” in *Alvey Vision Conference*, 1988.

[28] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov 2004.

[29] C. Ding, S. Fan, M. Zhu, W. Feng, and B. Jia, “Violence detection in video by using 3d convolutional neural networks,” in *Advances in Visual Computing* (G. Bebis, R. Boyle, B. Parvin, D. Koracin, R. McMahan, J. Jerald, H. Zhang, S. M. Drucker, C. Kambhamettu, M. El Choubassi, Z. Deng, and M. Carlson, eds.), (Cham), pp. 551–558, Springer International Publishing, 2014.

[30] M. Asad, Z. Yang, Z. Khan, J. Yang, and X. He, “Feature fusion based deep spatiotemporal model

for violence detection in videos," in *Neural Information Processing* (T. Gedeon, K. W. Wong, and M. Lee, eds.), (Cham), pp. 405–417, Springer International Publishing, 2019.

[31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[32] S. Sudhakaran and O. Lanz, "Learning to detect violent videos using convolutional long short-term memory," in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, 2017.

[33] A. Hanson, K. PNVR, S. Krishnagopal, and L. Davis, "Bidirectional convolutional lstm for the detection of violence in videos," in *Computer Vision – ECCV 2018 Workshops* (L. Leal-Taixé and S. Roth, eds.), (Cham), pp. 280–295, Springer International Publishing, 2019.

[34] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, (Cambridge, MA, USA), p. 568–576, MIT Press, 2014.

[35] Z. Islam, M. Rukonuzzaman, R. Ahmed, M. H. Kabir, and M. Farazi, "Efficient two-stream network for violence detection using separable convolutional LSTM," in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, jul 2021.

[36] Y. Su, G. Lin, J. Zhu, and Q. Wu, "Human interaction learning on 3d skeleton point clouds for video violence recognition," in *Computer Vision – ECCV 2020* (A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds.), (Cham), pp. 74–90, Springer International Publishing, 2020.

[37] S. Blunsden and R. Fisher, "The behave video dataset: ground truthed video for multi-person," *Ann. BMVA*, vol. 4, 04 2009.

[38] P. Rota, N. Conci, N. Sebe, and J. M. Rehg, "Real-life violent social interaction detection," in *2015 IEEE International Conference on Image Processing (ICIP)*, pp. 3456–3460, 2015.

[39] C.-H. Demarty, C. Penet, M. Soleymani, and G. Gravier, "Vsd, a public dataset for the detection of violent scenes in movies: design, annotation, analysis and evaluation," *Multimedia Tools and Applications*, vol. 74, 05 2014.

[40] M. Perez, A. C. Kot, and A. Rocha, "Detection of real-world fights in surveillance videos," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2662–2666, 2019.

[41] K. Yun, J. Honorio, D. Chattopadhyay, T. L. Berg, and D. Samaras, "Two-person interaction detection using body-pose features and multiple instance learning," in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 28–35, 2012.

[42] W. Sultani, C. Chen, and M. Shah, "Real-world anomaly detection in surveillance videos," 2019.

[43] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 1, pp. 172–186, 2021.

[44] J. Pedersen, N. Jensen, J. Lahrissi, M. Hansen, P. Staalbo, A. Wulff-Abramsson, and M. Sander, "Improving the accuracy of intelligent pose estimation systems through low level image processing

operations," in *International Conference on Digital Image & Signal Processing (DISP19)*, 04 2019.

[45] R. C. Gonzalez and R. Woods, *Digital image processing*. Upper Saddle River: Pearson, 4th ed. ed., 2018.

[46] M.-S. Kang, R.-H. Park, and H.-M. Park, "Efficient spatio-temporal modeling methods for real-time violence recognition," *IEEE Access*, vol. 9, pp. 76270–76285, 2021.

[47] NVIDIA, "Optical flow to RGB image algorithm." https://github.com/NVIDIA/flownet2-pytorch/blob/master/utils/flow_utils.py#L72. Accessed: 2022-05-06.

[48] F. Chollet *et al.*, "Keras." https://keras.io, 2015. Accessed: 2022-05-14.

[49] G. G. Cobo, "GitHub Pull Request: Solve memory inefficiency in RNNs." https://github.com/keras-team/keras/pull/16174, 2022. Accessed: 2022-05-14.

[50] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[51] "Kaggle." https://www.kaggle.com/. Accessed: 2022-05-14.

[52] OpenCV, "Histogram equalization tutorial." https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html. Accessed: 2022-05-06.

[53] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114, PMLR, 09–15 Jun 2019.

[54] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[55] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis* (J. Bigun and T. Gustavsson, eds.), (Berlin, Heidelberg), pp. 363–370, Springer Berlin Heidelberg, 2003.

[56] xlw1998, "GitHub Issue: The valid acc is 88.5 %,not 92 %." https://github.com/ahstarwab/Violence_Detection/issues/3, 2022. Accessed: 2022-05-14.

[57] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu, "RMPE: Regional multi-person pose estimation," in *ICCV*, 2017.

[58] Intel, "Openvino." https://docs.openvino.ai/latest/index.html#. Accessed: 2022-05-14.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID