VIOLENCE DETECTION IN VIDEOS USING 3D CONVOLUTIONAL NEURAL NETWORKS AND TRANSFER LEARNING

NAZ DÜNDAR

JULY 2023

ÇANKAYA UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF MATHEMATICS

M.Sc. Thesis in

MATHEMATICS

VIOLENCE DETECTION IN VIDEOS USING 3D CONVOLUTIONAL NEURAL NETWORKS AND TRANSFER LEARNING

NAZ DÜNDAR

JULY 2023

ABSTRACT

VIOLENCE DETECTION IN VIDEOS USING 3D CONVOLUTIONAL NEURAL NETWORKS AND TRANSFER LEARNING

DÜNDAR, NAZ

M.Sc. in Mathematics

Supervisor: Prof. Dr. Fahd JARAD

Co-Supervisor: Prof. Dr. Hayri SEVER

July 2023, 74 pages

Automatic violence detection using computerized systems instead of manpower has been a subject of significant contemporary interest among researchers recently. In addition, Deep Learning models such as Convolutional Neural Networks have been successfully applied to many different tasks in a wide range of domains, including video recognition. To that end in this thesis, a computerized model for violence recognition will be designed which does not require manual human inspection. Two models will be designed, including a simple 3D CNN and a MoViNet 3D CNN which uses transfer learning. A combined dataset consisting of 5200 videos will be used to train and run the models. The aim of this thesis is to give a comprehensive explanation to the design and mathematics of CNNs, implement two 3D CNN models and explain and analyze them in many aspects.

Keywords: Violence detection, Deep Learning, Convolutional Neural Networks, Transfer learning.

ÖZET

EVRİŞİMSEL SİNİR AĞLARI VE TRANSFER ÖĞRENME İLE VİDEOLARDA TEHLİKE TESPİTİ

DÜNDAR, NAZ

Matematik Yüksek Lisans

Danışman: Prof. Dr. Fahd JARAD

Ortak Danışman: Prof. Dr. Hayri SEVER

Temmuz 2023, 74 sayfa

İnsan gücü yerine bilgisayarlı sistemlerin kullanıldığı otomatik şiddet tespiti son zamanlarda araştırmacıların ilgi konusu olmuştur. Ek olarak, Evrişimli Sinir Ağları gibi Derin Öğrenme modelleri, video tanima da dahil olmak üzere çeşitli alanlarda birçok farklı göreve başarıyla uygulanmıştır. Bunlar göz önünde bulundurularak bu tezde, şiddetin tanınması için manuel insan kontrolü gerektirmeyen bilgisayarlı bir model tasarlanacaktır. Basit bir 3D CNN ve transfer öğrenme kullanan bir MoViNet 3D CNN dahil olmak üzere iki model tasarlanacaktır. Modelleri eğitmek ve çalıştırmak için 5200 videodan oluşan birleştirilmiş bir veri kümesi kullanılacaktır. Bu tezin amacı, CNN'lerin tasarımı ve matematiği hakkında kapsamlı bir açıklama sağlamak, iki 3D CNN modelini implemente etmek ve bu modelleri birçok yönden açıklamak ve analizini yapmaktır.

Anahtar Kelimeler: Tehlike tespiti, Derin Öğrenme, Evrişimsel Sinir Ağları, Transfer öğrenme.

ACKNOWLEDGEMENTS

I would like to thank my family Dafne, Murat, Onur and Yiğit for their continuous support of all kinds, my friends and colleagues, my supervisors Fahd Jarad and Hayri Sever for their guidance and encouragement, Ayhan Arıcı for all the late night study sessions, all of the teachers I was lucky enough to be the students of, Çankaya University for the incredible opportunities they have granted me and TÜBİTAK for their financial support. I am eternally grateful for all of you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

SYMBOLS

$\star$	:Cross-Correlation
$*$	:Convolution
σ	:Standard Deviation
η	:Learning Rate
∇	:Gradient Operator
∂	:Partial Derivative Operator

ABBREVIATIONS

AI	:Artificial Intelligence
ML	:Machine Learning
BoW	:Bag-of-Words
SVM	:Support Vector Machine
STIP	:Space-Time Interest Points
MoSIFT	:Motion Scale-Invariant Feature Transform
ViF	:Violent Flows
ANN	:Artificial Neural Network
CNN	:Convolutional Neural Network
LSTM	:Long Short-Term Memory
DNN	:Deep Neural Network
ConvLSTM	:Convolutional Long Short-Term Memory
CPU	:Central Processing Unit
GPU	:Graphics Processing Unit
ReLU	:Rectified Linear Unit function
BP	:Backpropagation Algorithm
MoViNet	:Mobile Video Network
NAS	:Neural Architecture Search
SE	:Squeeze-and-Excitation

CGAP	:Cumulative Global Average Pooling
FLOP	:Floating Point Operation
TP	:True Positive
TN	:True Negative
FP	:False Positive
FN	:False Negative
tanh	:Hyperbolic Tangent Function

CHAPTER I
INTRODUCTION

Over the past few decades, the application of Machine Learning (ML) models has gained significant traction across a wide range of industries and disciplines, increasingly replacing the need for humans in certain tasks. The problem of violence detection using automated systems is one of the many practical real-life applications of ML. The context of this problem aims to decreace reliance on human-operated surveillance systems, driven by the pursuit of finding more efficient and effective alternatives to detect violence in public places. Equipping surveillance systems with ML models for violence detection has the potential to enhance public safety. The aim of this thesis is to contribute to the development of more efficient and accurate computerized violence detection systems capable of being deployed in diverse real-world scenarios by exploring benchmark ML algorithms and proposing alternative models.

1.1 MOTIVATION AND PROBLEM DEFINITION

There has been a significant increase in the amount of public violence worldwide in the recent years. For example, according to publicly available statistics, violent crime rates were at their peak in 1990s and were displaying a decrease up until 2010s in the United States of America. Then especially around 2014, the rates began to increase again, almost doubling the amount recorded just before 2014. There are many factors believed to be contributing to the recent increase, including but not limited to psychological and physiological effects of COVID, worldwide economic contraction, advances in technology, political views, local and global awareness of police brutality leading to less use of force by the police, corruptions in the justice system, etc. These reasons apply not only to the United States, but all countries. With the rise of public violence statistics, problems emerged about keeping public places safe and secure. This concern led to the installation of surveillance cameras in several public scenes.

Surveillance cameras are useful for monitoring public places, detecting and identifying any anomalies such as violent behavior, which allows the authorities to take the necessary actions. However, their effectiveness is questionable, because they require constant manual human supervision. In other words, identifying a scene as violent or not is solely dependent on the supervisor's skills, carefulness and judgment. The human decision-making process is usually slow and biased. Humans are mostly incapable of monitoring simultaneous stimulations which would be required in the presence of multiple cameras. Since increasing the number of people inspecting the cameras directly increases the manpower expenses, it is not an ideal solution. By any means, any amount of manpower is expensive overall. Moreover, humans can easily lose focus at times, make biased or wrong predictions or decisions. Additionally, most security cameras are not inspected constantly, only when needed. Thus, it is clear that human dependency of surveillance systems is ineffective, impractical and highly insufficient, especially regarding security concerns. In order to overcome the limitations of human involvement and allow surveillance systems to detect violent acts more effectively, researchers began developing computational systems where human supervision would not be required. The aim is to build low-cost, computer-driven violence detection models that can be integrated into surveillance systems which will enable uninterrupted human-independent supervision while raising minimum amount of false alarms.

The goal of this work is to develop a computerized model that is capable of detecting violent behavior in videos. Violence detection refers to the act of detecting and classifying violent, intrusive and hostile behavior present in input data. In order to develop an extensive violence detection model, the scope of violence must be well-defined. The extent of a violent act may vary from a two-person physical fight to a person pointing a gun, mass shootings, home invasion, stabbing, terror attacks, etc. It is also important to note that most surveillance systems do not have audio features. To that end, two end-to-end, trainable ML models, 3D Convolutional Neural Networks (CNNs), are proposed in this work. One of the models will be built from scratch, the other model will use transfer learning with MoViNets [55] and the results will be compared and discussed in detail. No audio features will be used, only visual video footage. The extent of violence in this work is defined as an aggressive invasion of personal space or a physical fight between two or more persons. It must be noted that

the scope of violence is neither limited to this definition nor limited to humans, but this is the definition of violence that will be used in this work.

1.2 THESIS ORGANIZATION

This thesis is organized as follows: A literature review on automatic violence recognition models and a detailed information on CNNs including their working principles, designs, algorithms and mathematical overview are presented in Chapter II. A comprehensive explanation of the proposed models is given in Chapter III. The experimental setups, results and model evaluations of the experiments conducted for both models are discussed in Chapter IV. An overall recap and potential future works are presented in Chapter V, where the thesis is concluded.

CHAPTER II
LITERATURE REVIEW AND BACKGROUND

In this chapter, a detailed literature review of the previously proposed violence detection models as well as their evaluations and essential information on Deep Learning (DL) algorithms, mainly focusing on CNNs, are presented.

2.1 RELATED WORKS

Researchers have been using computer vision algorithms for pattern recognition tasks such as violence detection for over two decades. These models vary from traditional hand-crafted models to modern deep learning algorithms. In this section, a detailed literature review of both hand-crafted and DL methods for violence detection will be presented as well as their efficiencies, accuracies and limitations.

2.1.1 Hand-Crafted Methods

Former studies on violence detection in videos generally used visual and/or audio features to detect flame and blood [38], skin and blood [41], gunshots and explosions using Gaussian mixture models and Hidden Markov Models [40], etc. Later, the Bag-of-Words (BoW) procedure, often used for images, was adapted to videos [39] and was used frequently for video classification tasks. For example, [42] used spatio-temporal video cubes and the BoW approach for aggressive behavior detection. [1] developed a method for verifying person identity and detecting unusual human behavior based on the descriptors derived from Histograms of Optical Flow at the automated Access Control Points. Their method used normalized Levenshtein distance [43] to detect similarities between two motion sequences. A Gaussian Model of Optical Flow was used in [6] to detect not only violence, but also the location of the act as seen in the video. They also proposed a novel descriptor, Orientation Histogram of Optical Flow, to differentiate between violent and non-violent behavior, which are then fed to a linear Support Vector Machine (SVM) for classification. Their proposed model was tested on CAVIAR [4], and reached an accuracy of 86.75%.

BoW framework was tested in [16] along with action descriptors Space-Time Interest Points (STIP) [10] and Motion Scale-Invariant Feature Transform (MoSIFT) [8]. They also created the Hockey Fight Dataset [16] and Movies Dataset [16] which are frequently used in violence detection experiments. They reached an accuracy of 89.50%. The method developed in [18] considered statistics of how flow-vector magnitudes change over time and these statistics are represented using the Violent Flows (ViF) descriptor. ViF descriptors are then classified using linear SVM. The highest accuracy they received is 82.90%. [19] also employed MoSIFT algorithm and used Kernel Density Function to eliminate the feature noise. They reached an accuracy of 89.05%. [20] analyzed the features of the motion vectors in each frame and between the frames and got Region Motion Vectors descriptor. They used SVM for classification. They also created the VVAR10 [20] dataset. Their proposed method reached an impressive accuracy of 96.10%. ViF descriptor was used with Horn-Schunck [17] for violence detection in [21]. Then, they applied the non-adaptive interpolation super resolution algorithm to improve the video quality and fire Kanade-Lucas-Tomasi face detector. They used the BOSS dataset for experiments and reached an accuracy of 97.00%.

2.1.2 Deep Learning Methods

3-dimensional Convolutional Networks (3D ConvNets) are used in [22] for spatio-temporal feature learning. They used the UCF101 dataset [12] for model evaluation and reached an accuracy of 90.40%. The AlexNet Model [15] pre-trained on the ImageNet database [15] was used in [23] as the CNN model for extracting features. The extracted features are then aggregated using the Convolutional LSTM (convLSTM) layer of their proposed architecture. They used Hockey Fights Dataset [16], Movies Dataset [16] and Violent-Flows Crowd Violence Dataset [18] in their experiments and reached an accuracy of 91.10% on the Hockey dataset and 100.00% on the Movies dataset [16]. A modified 3D ConvNet framework was proposed in [47], which improves the preprocessing method of 3D ConvNet. Video sequence is cut into clips based on key frames which decreases the motion integrity loss and redundancy caused by uniform sampling. Their method was simple yet effective, reaching impressive accuracies of 99.62% on Hockey Fights and 99.97% on Movies datasets. A combination of CNN and Long-Short Term Memory (LSTM) was used in [24] for spatial feature extraction and classification, respectively. They used the Hockey

Dataset [16] and achieved a 98.00% accuracy. [25] used spatio-temporal features with 3D CNN for prediction of violent activity. They tested their model on three datasets and reached 99.90% accuracy on the Movies Dataset [16]. [26] used VGG-16 [11], pre-trained on ImageNet [15], as spatial feature extractor followed by LSTM as temporal feature extractor and sequence of fully-connected layers for classification. They also introduced a new dataset called Real-Life Violence Situations [26], which they used to test their model and reached an accuracy of 88.20%.

Three ImageNet [15] models were used for feature extraction which are then fed to an LSTM network in [27]. They created a new dataset in Bangladesh context. They were able to reach an accuracy of 97.06%. In order to reduce feature redundancy with no extra parameters, [28] proposed compact convolution. The proposed method was used for image classification. A combinition of 3D CNN and SVM was used in [48]. The network was pre-trained on the Sport-1M dataset [61] and used for feature extraction. The output was then fed as an input to a classifier, which is a linear SVM in the case of binary classification. [30] proposed a novel violence detection pipeline that can be combined with 2D CNNs. They also presented a spatial attention module called Motion Saliency Map (MSM) and a temporal attention module called Temporal Squeeze-and-Excitation (T-SE) to improve the performance of violence detection. They tested their models on five datasets and reached an accuracy of 100.00% the highest and 92.00% the lowest. [29] used a combination of Xception [14], pre-trained on the ImageNet dataset [15], and LSTM for feature extraction and classification, respectively. They used three datasets for evaluation and their highest accuracy recorded was 98.32%. A novel architecture of end-to-end CNN-LSTM model was proposed in [31] that could run on low-cost Internet of Things devices. The model was tested on two datasets and achieved an average accuracy of 73.35%.

A two-stream DL architecture is proposed in [32] leveraging Separable Convolutional LSTM (SepConvLSTM) and pre-trained MobileNet [13]. They used three datasets for model evaluation and the highest result they achieved was 99.50%. An approach that combined VGG-16 [11] model pre-trained on ImageNet [15] with ConvLSTM [18] was proposed in [33] for detecting violence in surveillance video datasets. Their model was tested on six datasets and reached an accuracy of 100.00% the highest and 92.40% the lowest. [34] designed a model where the well-known CNN, ResNet50 [9], was used for feature extraction followed by ConvLSTM for detecting anomalies. They used the UCF-Crime dataset [5] and achieved an accuracy of 81.71%.

A combination of 3D CNN and CNN Bidirectional LSTM (CNN-BiLSTM) was proposed in [35]. They used three datasets for their experiments and the highest accuracy they achieved was 94.90%. [36] proposed two methods, 3D DenseNet Fusion OF RGB and 3D DenseNet Fusion OFnom RGB, and developed a new dataset called AICS-violence. The highest accuracy they achieved was 97.675%. [52] proposed a U-Net-like network that uses MobileNet V2 for feature extraction followed by an LSTM for temporal feature extraction and classification. [53] also used a MobileNet deep learning model for real-time violence detection in surveillance videos, which reached 96.66% accuracy.

2.2 BACKGROUND

This section provides comprehensive background information on the field of Artificial Intelligence (AI) and the development of Artificial Neural Networks (ANN) as well as an in-depth overview of CNNs.

AI is a branch of computer science that is concerned with developing computational systems that are capable of performing tasks usually within the capabilities of humans. It refers to the intelligence of machines where they are able to experience, perceive, analyze, learn and speculate information in a way human intelligence does, based on some form of raw data. This data can be textual, audio or visual data. Intelligent machines are created in such a way that imitate human cognitive abilities. AI systems can use the learned information and experiences to self-improve their performance over time. AI can be split into two subcategories as Narrow AI and General AI. Narrow AI, also known as Weak AI, is used in particular, focused tasks and is the kind of AI we use in real-life applications. Digital assistants in mobile devices, search engines like Google, self-driving vehicles, robotics and online games played against the computer are some examples of the applications of Narrow AI. The abilities of Narrow AI are limited to the task-at-hand and its respective domain and these kinds of systems need to learn from thousands, even millions of labeled information or examples. The learned knowledge usually can not be transferred to other tasks or domains. General AI, on the other hand, is promised to perform human-level intelligent action, have a full range of human cognitive abilities, learn from a small number of examples as well as unstructured data and the possibility of transferring the learned information to other tasks within the same or other domains. However, the development of General AI remains a challenge and mostly theoretical

for the time being. Even though General AI is not yet applicable with the technology we have today, Narrow AI manages to cover an impressive scope of tasks and domains.

Computer vision is a field of computer science and AI that is concerned with enabling computers to learn low to high-level features, i.e extract information from a given input in the form of a text, image or video. This is achieved by developing methods and algorithms to teach computers to understand and analyze visual data, similar to the way humans understand and interpret the same kinds of data. The input, in the form of visual data, is processed and analyzed and a decision or prediction is made based on the learned information. In other words, the idea behind computer vision algorithms is to build computational models that are able to perform tasks that the human visual system can do at a human-level accuracy and precision. Computer vision can be divided into specific domains such as pattern recognition, image reconstruction, object detection, motion estimation, 3D reconstruction, etc. It was developed a few years after the concept of AI emerged. Although these disciplines have been around for over half a century, the recent technological advancements and the availability of an enormous amount of data have certainly affected the development of computer vision systems positively. Computer vision, overall AI, is now studied and applied in a wide range of fields including medicine, engineering, economics, etc.

Machine learning (ML), a branch of AI, is a broad field of study concerned with understanding and building models that are able to take an input of some form, extract high-dimensional information from the input, learn from the extracted features and make predictions or decisions based on the learned information. Similar to the learning process humans go through, they learn from data and improve their performance accordingly. ML algorithms are commonly used for imitating human activities. At their core, ML algorithms are mathematical functions that represent the relationship between different features of data. Like any mathematical function, they map certain variables given a dataset, to a target variable. For example, in the case of an image classification model, the input images are mapped, thus classified, to their respective labels. Therefore, learning algorithms are also mathematical functions with the purpose of finding an optimizing function through the training process with respect to a certain set of parameters which minimizes loss over a given dataset.

2.2.1 Artificial Neural Networks

There are billions of neurons in the biological brain that allows for the connection of interaction and the resulting behavior. The most important distinction of the animal brain is its ability to learn. Animals learn from patterns, which can be experienced through vision, hearing, touch, taste or smell. All animals possess the ability to leverage learning, albeit minimum amounts, as one of the most primal survival skills. By all means, this ability is far more advanced in human brains. Humans can learn to talk, remember faces they have seen, process and memorize information, etc. The learning taking place in the brain is executed by densely interconnected neural networks. Neurons, the information messengers of the brain, communicate by sending electrical and chemical signals through the nervous system.

ANNs are computational processing systems loosely inspired by the biological neural networks. In comparison with the biological brain morphology, ANNs have a small number of hundreds or thousands of processor units [58]. They consist of artificial neurons, also called nodes, which are interconnected and are able to process inputs and transmit signals to other neurons. The connection of neurons are called edges, resembling synapses in the biological brain. Neurons and edges usually have a preassigned weight, which is updated in the process of learning. In this context, learning refers to the process of hidden layers making decisions or predictions based on the output of the previous layers. Artificial neurons are mathematical functions that are used to calculate the weighted sum of the inputs and give the output in the form of an activation map. Activation maps specify the significant features of the input. In other words, these neurons collaborate in a distributed fashion to collectively acquire knowledge about features and patterns from the input in order to optimize the final output [37].

The basic architecture of an ANN consists of an input layer, one or more hidden layers and an output layer, where layers are composed of artificial nodes. In a fully-connected ANN, as the name suggests, each node within a layer is connected to every node in the preceding and subsequent layers. The diagram of a basic ANN is given in Figure 1. The size and computational complexity of ANNs are relatively small, thus they can be implemented on a Central Processing Units (CPUs), which is the main processor of any computer capable of executing arithmetic, logic and standard input/output operations.

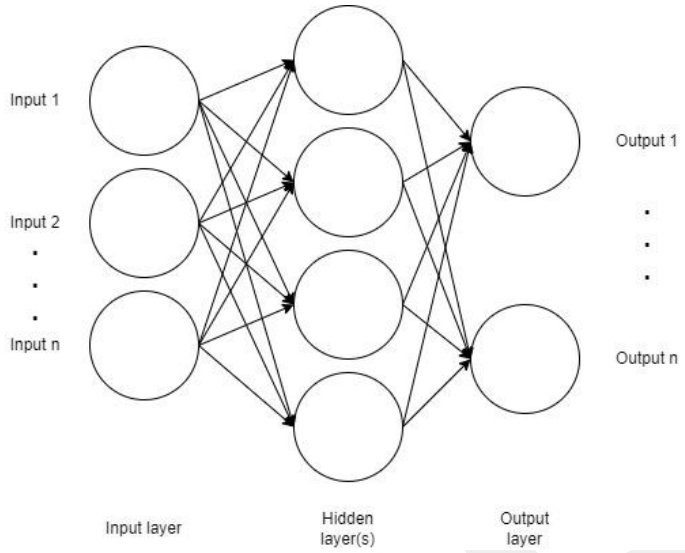


Figure 1: The architecture of a basic ANN model

ANNs work well with simple tasks such as handwritten digit classification, like the MNIST database [54]. So, what would happen if an ANN was given a task with higher complexity? Intuitively, the number of neurons or layers can be increased accordingly in order to work with more difficult data. However, as the number of neurons and layers increase, so do the number of mathematical operations and therefore the complexity of the network. The model would require higher resources such as computational power, time and memory to store the data, which are usually limited. Also, as the number of parameters increase, the network would be more prone to overfitting. As the name suggests, overfitting occurs when a network classifies the training data too well and fails to model data it has not been trained on, in other words, is unable to learn effectively. Overfitting is a common and important issue for all ML models and will be further explained later in this chapter. So, if the number of neurons or layers in an ANN were simply increased, overfitting would be difficult to reduce. Thus, more effective models with fewer parameters were needed for more difficult tasks.

2.2.2 Convolutional Neural Networks

Deep Neural Networks (DNNs) are a type of ANNs, composed of multiple hidden layers. The depth of a neural network refers to the number of hidden layers of the network. DL, a subset of ML, refers to the study of DNNs, which are ANNs at their core, only with a higher number of hidden layers. The basic building blocks of DNNs are the same as ANNs, which are nodes, edges, weights, biases and non-linear

functions. They can be trained using the same training algorithms as ANNs. DNNs are capable of processing and modeling non-linear relationships with higher complexities in comparison to ANNs. As the number of layers, therefore number and complexity of mathematical operations, increases, higher computational power is demanded. To that end, they can be implemented on both CPUs and Graphics Processing Units (GPUs). However, due to their high computational complexity, their training on CPUs take longer than ANNs. Although GPUs were not specifically designed for the implementation of DNNs, they are a favorable substitution for CPUs. DNNs are more capable of discovering multiple levels of representation and extracting higher-level features than simple ANNs. Low-level features can be considered as lines and edges whereas high-level features as numbers and faces. Overall, DNNs are usually preferred to ANNs for pattern recognition tasks involving complex, high-dimensional features.

CNNs were first developed and introduced in the 1980s by Yann LeCun. CNNs are a special type of DNNs in the DL field generally used in image or video pattern recognition tasks. The emerging of CNNs revolutionized the analysis of high-dimensional data existing in many different forms, such as text, audio, image and video, which are generally difficult to store and manage. CNNs are structurally similar to DNNs, but they differ from DNNs by the use of the convolution operation. CNNs are capable of learning to optimize the kernels, thus they use less pre-processing than other traditional algorithms. CNNs are a type of feed-forward neural networks, which means the information only moves in one direction, forward, starting from the input nodes, through the hidden nodes and to the output nodes. They use supervised learning, which is when pre-labelled inputs are leveraged to train algorithms. The goal of supervised learning is to reduce the overall classification error of the network [37]. Most image-driven pattern recognition models use supervised learning. CNNs are rate-based neural networks, which means that they are suitable for implementation on conventional CPUs with substantial numerical processing capabilities. However, CNN algorithms have grown more intricate and thus require more powerful computing platforms such as GPUs [2].

Digital images are stored as 2-dimensional pixel matrices. The dimension of an image is represented by $H \times W$ where H denotes the height dimension and W denotes the width dimension. H is equal to the number of pixels across the height and W is equal to the number of pixels across the width of the image. For example, in grayscale images, which are black and white images, the pixels are assigned numerical values

ranging from 0-255 which represent the intensity of the pixels. 0 denotes black, 255 denotes white and the numbers in between whiten in shade starting from 0 as the value converges to 255. Another example of images is a color image, called RGB in literature, which is comprised of a matrix of pixels in three dimensions. For RGB images, a third dimension of depth is added.

2.2.2.1 Architecture of CNNs

A typical CNN consists of three types of layers, which are convolutional layers, pooling layers and fully-connected layers. The input of a CNN undergoes several layers of floating point operations and matrix computations. Since CNNs are a type of fully-connected neural networks, each node in a layer is connected to each of the nodes in the former and latter layers. When the CNN is fed an input image in the form of a number matrix, each layer of the CNN generates diverse activation maps. These activation maps specify the significant features of the input image. Earlier layers focus on simple features, such as colors and edges. As the data proceeds through the latter layers of the network, it begins to recognize more complex features of the data until it eventually makes a prediction [3]. Each neuron in a CNN architecture usually takes the input in the form of a matrix, takes the product of their values and their relative assigned weights, adds them up and gives them as input to their corresponding activation function. The output of each layer is the input of the subsequent layer.

2.2.2.1.1 Input Layer

The input layer is trivially the first layer of a CNN. In the case of images, the pixel matrix of the input image is fed in the input layer.

2.2.2.1.2 Convolutional Layer

The convolutional layer is the first layer in a CNN architecture where extraction of features and where most of the computation of the model occurs. A convolutional layer demands three elements, which are an input image, a filter, and an activation (feature) map. As the name suggests, this layer performs convolution. In Mathematics, convolution operation shows how the shape of one function is influenced by another function. It takes two functions as input and produces a third function as output. In neural networks, convolution operation refers to the act of sliding a matrix called a kernel or a filter of a specified size $K \times K$ with learnable weights across the input

image, which is also a matrix as mentioned above. The kernel is slid across by a predefined stride, which is the step size for each step of the sliding action, and this process is repeated until the whole image is swept by the kernel. The scalar (dot) product is calculated between the kernel and the parts of the input image based on the size of the kernel. So, the output of a convolutional layer is the weighted sum of input and weights, also known as activation map. The kernel must be rotated 180° counterclockwise to perform convolution on images since convolution is implemented using a digital filter. If the kernel is used without being rotated first, the process is called cross-correlation. In other words, convolution is the same operation as cross-correlation with the key difference of rotating the kernel 180° counterclockwise. This can be further explained mathematically as Equation 2.1 where $*$ represents convolution and $\star$ represents cross-correlation.

$$f * g = f \star rot180(k) \tag{2.1}$$

2.2.2.1.2.1 Hyperparameters of a Convolutional Layer

There are certain variables that determine the structure of a neural network and how it is trained. These variables are called hyperparameters. Hyperparameters are tuned independently from the model parameters and are usually defined before the learning process begins [44]. The hyperparameters of a convolutional layer are number of filters, kernel size, padding and stride.

2.2.2.1.2.1.1 Number of Filters

Number of filters in a convolutional layer refers to the depth of the layer. It determines how many distinct features the model can learn to detect from the input. In other words, these learnable filters will learn to activate for different features of the input. It is set based on several factors such as the complexity of the model or data and the task-at-hand. It is usually defined through fine-tuning or experimentation. Choosing a too small value might result in underfitting, which means the model would fail to catch certain patterns in the input. On the other hand, choosing a too large value might result in overfitting, which means the model learns the training data too well and therefore fails to generalize to other data.

2.2.2.1.2.1.2 Kernel Size

Kernel size, also called receptive field or filter size, refers to the spatial dimensions of the kernels that will be used for convolution operations. In 2D CNNs, kernel size is comprised of spatial dimensions width and height, because the kernel moves and convolves in 2 dimensions. However, in 3D CNNs, kernel size is comprised of depth, width and height, because the kernel moves and convolves in 3 dimensions. Small kernel sizes are used for local features and large kernel sizes are used to capture global features. The increase in the size of the kernel results in an increase in the computational complexity of the model.

2.2.2.1.2.1.3 Padding

Convolution results in a shrinkage in the dimensions of the input data. This may cause important spatial information near the edges to be lost after convolution. In order to prevent it, padding is applied on the input data, which refers to the operation of adding a border of zeros around the input before convolution occurs. With the use of padding, the dimensions of the input is preserved and the output feature maps are prevented from being smaller than input data. There are two main types of padding used in CNNs, which are same padding and valid padding. Same padding is when zeros are added around the borders of the input data. Valid padding refers to no padding being applied to the input. The type of padding to be used can be determined based on the complexity of the data being used.

2.2.2.1.2.1.4 Stride

The stride hyperparameter is used to control the step size of the kernel. In other words, it determines the number of spatial units (pixels) the kernel will be slid horizontally and vertically at each step during convolution. A small stride results in a larger feature map with preserved spatial dimensions whereas a large stride results in a smaller feature map with reduced spatial dimensions. Intuitively, smaller stride can be used when working with immensely detailed data when details are more important than reducing computational cost of the model. However, if spatial information loss to some extent is not crucial to the task-at-hand, a larger stride can be used instead, in order to decrease the number of matrix operations and thus reduce the model complexity, size and run-time.

The output of a convolutional layer depends on the size of the input image, size of the kernel, stride and padding. For a grayscale image of dimensions $H \times W$ swept with a kernel of dimensions $K \times K$, with stride S and padding P, the output of the convolutional layer is calculated as given in Equation 2.2 where H' and W' are the height and width of the output, respectively. The output dimensions are expected to be integers, and if not, it means that stride was not defined correctly.

$$H' = \frac{H-K+2P}{S} + 1$$
$$W' = \frac{W-K+2P}{S} + 1 \tag{2.2}$$

For example, assume that a 4 × 4 input matrix is convolved with a 2 × 2 kernel with stride of 1 and no padding. The visualisation of the operation is given in Equation 2.3. The dimensions of the resulting matrix is 3 × 3, which can be determined using Equation 2.2. The scalar product between the receptive field of the input and the kernel can be calculated using Equation 2.4 where $a_{i,j}$ represents the element of the resulting matrix in the *i*th row and *j*th column starting from index 0 up to index 2.

$$\begin{bmatrix} 0 & 1 & 2 & 1 \\ 3 & 2 & 2 & 0 \\ 2 & 1 & 1 & 1 \\ 0 & 0 & 1 & 3 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 2 \\ 4 & 3 & 3 \\ 2 & 2 & 4 \end{bmatrix} \tag{2.3}$$

$$\begin{aligned} a_{0,0} &= 0(1) + 1(0) + 3(0) + 2(1) = 2 \\ a_{0,1} &= 1(1) + 2(0) + 2(0) + 2(1) = 3 \\ a_{0,2} &= 2(1) + 1(0) + 2(0) + 0(1) = 2 \\ a_{1,0} &= 3(1) + 2(0) + 2(0) + 1(1) = 4 \\ a_{1,1} &= 2(1) + 2(0) + 1(0) + 1(1) = 3 \\ a_{1,2} &= 2(1) + 0(0) + 1(0) + 1(1) = 3 \\ a_{2,0} &= 2(1) + 1(0) + 0(0) + 0(1) = 2 \\ a_{2,1} &= 1(1) + 1(0) + 0(0) + 1(1) = 2 \\ a_{2,2} &= 1(1) + 1(0) + 1(0) + 3(1) = 4 \end{aligned} \tag{2.4}$$

2.2.2.1.3 Pooling Layer

In a typical CNN architecture, a convolutional layer is usually followed by a pooling layer. Pooling operation is similar to convolution. Pooling groups up the pixels in the input image and filters them down to a subset. A kernel is slid over the output of the convolutional layer, which is a feature map, preceding the pooling layer. The kernel calculates an output on the receptive field, which is the region in the image that a specific feature is looking at. The aim of the pooling layers is to further reduce the computational complexity and cost of the presented model. The pooling layers aim to decrease the size of the convolved feature map by decreasing the connections between layers. Pooling layers are also used to summarize the features present in a region of the feature map produced by a convolutional layer. This is done for the purpose of down-sampling.

There are different types of pooling, but the type usually used for image or video classification tasks is max pooling. In max pooling, the kernel picks the maximum pixel value in the receptive field. For example, a kernel of size 2×2, the receptive field has 4 pixel values, and the maximum value of those 4 values is selected. The hyperparameters of the pooling layer are kernel size and stride, which refer to the spatial dimensions of the kernel that will be used in the pooling operation and step size of the kernel movements, respectively. Together, they determine the amount of down-sampling that will be performed in the layer.

2.2.2.1.4 Batch Normalization

A normalization layer is optional in CNNs. Normalization is commonly used for standardizing raw data to downscale the range in which the data exists. It increases the learning rate and convergence speed of the model, prevents model divergence and thus, makes it easier to train [44]. Batch normalization [50] is a type of normalization. Instead of in the raw data, batch normalization is done between the layers of the model along training mini-batches and can be added as a layer itself. Batch normalization allows us to use much higher learning rates and be less careful about initialization, and in some cases eliminates the need for Dropout [50].

2.2.2.1.5 Fully-Connected Layer

The Fully-Connected layer consists of neurons where each neuron receives input from all the neurons in the previous layer, also referred to as the Dense layer.

There is usually a Flatten layer preceding the Dense layer, because neural networks accept the input as a 1-dimensional linear vectors. Flatten layer, as the name suggests, flattens the output matrix of the preceding layer into a 1D vector. After the features are extracted in previous layers, they are classified in the Dense layer based on the output of the convolutional layers.

2.2.2.1.6 Output Layer

The output layer is the last layer of the network. It produces the desired prediction or classification of the network.

2.2.2.2 2D CNNs

The term CNN usually refers to the standard type, which is 2 dimensional CNN. It is called 2D, because the kernel slides in two dimensions on the data. Images only have spatial information, which refers to the height and width dimensions. 2D CNNs are generally applied to image data and they are usually difficult to be outperformed in image classification tasks. 2D convolution is carried out to extract features from local neighborhood on feature maps in the previous layer. After a bias term is added, the result is passed through a sigmoid function [49]. Mathematical representation of the 2D convolution operation is given in Equation 2.5, where K is the convolution kernel, A is the convolution matrix and B is the resulting matrix [51].

$$B(i,j) = \sum_{m=0}^{M} \sum_{n=0}^{N} K(m,n) * A(i-m, j-n) \tag{2.5}$$

The formal equation for the value of a unit at position (x, y) in the jth layer denoted by v_{ij}^{xy} was defined by [49], which is given in Equation 2.6, where tan$h(\cdot)$ is the hyperbolic tangent function, b_{ij} is the bias term for the current feature map, m is the index over the set of feature maps in the $(i-1)$th layer are connected to the feature map at the current step, w_{ijm}^{pq} is the value at the position (p, q) of the kernel which is connected to the kth feature map, and P_i is the height and Q_i is the width dimensions of the kernel. A basic 2D CNN architecture is given in Figure 2.

$$v_{ij}^{xy} = \tanh\left(b_{ij} + \sum_{m} \sum_{p=0}^{P_i-1} \sum_{q=0}^{Q_i-1} w_{ijm}^{pq} v_{(i-1)m}^{(x+p)(y+q)}\right) \quad (2.6)$$

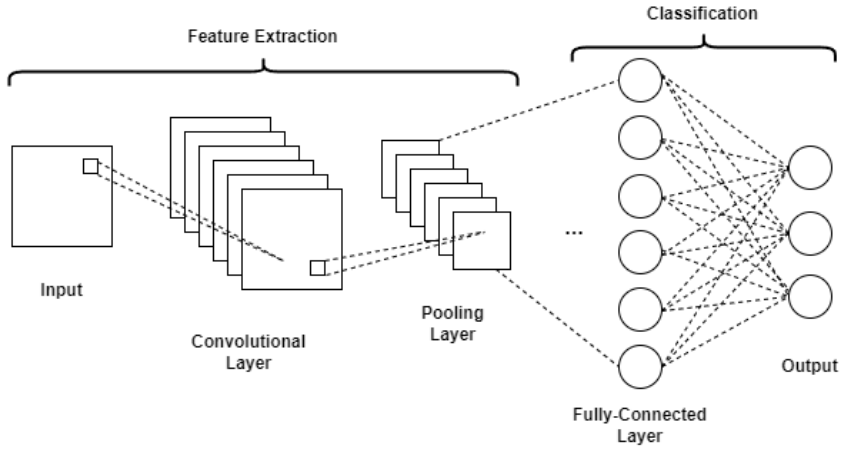


Figure 2: The architecture of a basic 2D CNN model

2.2.2.3 3D CNNs

Although 2D CNNs achieve outstanding results with image data, they are limited to dealing with spatial information. So, a 2D CNN is insufficient when it comes to working with video data. The definition of a video sequence is a series of images shown in rapid succession to give the impression of continuous motion [47]. In other words, videos are essentially consecutive images, which have spatial features. However, there is also motion information, also called temporal information, present between adjacent frames of a video, which would simply be lost if a 2D CNN is used on video data. Changes in successive frames have crucial effects on the results of a video classification problem. In order to deal with the temporal features that happen over time in a video, a third dimension of *time* must be added to a 2D CNN.

3D convolution is the same operation as 2D convolution, except the kernel slides in 3 dimensions in 3D convolutions as opposed to 2 dimensions in 2D convolutions. 3D convolution is obtained using a 3D kernel on the cube formed by stacking adjacent frames together [48]. Thus, movement information is acquired from consecutive frames. So, by using 3D convolution and 3D pooling, temporal information of the input video remains well preserved [47]. Mathematical representation of 3D convolution is similar to that of 2D convolution and is given in Equation 2.7, where K is the convolution kernel, A is the convolution matrix and B is the resulting matrix [51].

$$B(i,j,r) = \sum_{m=0}^{M} \sum_{n=0}^{N} \sum_{t=0}^{T} K(m,n,t) * A(i-m, j-n, r-t) \qquad (2.7)$$

The formal equation, again defined by [49], for the value of a unit at position (x, y, z) in the jth feature map in the ith layer denoted by v_{ij}^{xyz} , is given in Equation 2.8 where tan$h(\cdot)$ is the hyperbolic tangent function, b_{ij} is the bias term for the current feature map, m is the index over the set of feature maps in the $(i-1)$th layer which are connected to the feature map at the current step, w_{ijm}^{pqr} is the (p, q, r)th value of the kernel which is connected to the mth feature map in the preceding layer, P_i is the height and Q_i is the width of the kernel, and R_i is the size of the 3D kernel across the temporal axis. By this construction, the feature maps in the convolution layer are connected to multiple contiguous frames in the previous layer, thereby capturing motion information [49]. A typical 3D CNN architecture with two output classes is given in Figure 3.

$$v_{ij}^{xyz} = \tanh\left(b_{ij} + \sum_{m} \sum_{p=0}^{P_i-1} \sum_{q=0}^{Q_i-1} \sum_{r=0}^{R_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)}\right) \qquad (2.8)$$

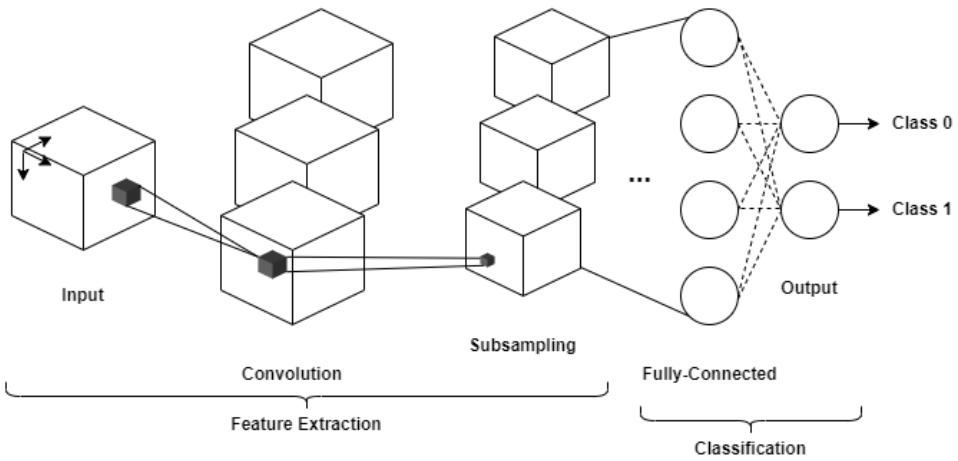


Figure 3: The architecture of a typical 3D CNN model

2.2.3 Activation Functions

Activation functions are mathematical functions used in ANNs that determine whether a neuron in the network structure should be activated or not, and if activated,

to what degree. They are applied to each neuron in the hidden layers of the network after the weighted sum of the set of input values is calculated to get the output values of the layer. In a nutshell, they are used to transform input signals to output signals which are to be fed to the subsequent layer as input. Any perceptron, a simple artificial neuron that has one input layer and one output layer, contains a summation function and an activation function. The inputs fed to a perceptron are processed by the summation function followed by the activation function to give the output. There are different types of activation functions, each serving a specific purpose. They must be chosen carefully based on the task-at-hand. Choosing the most suitable activation function directly increases the prediction accuracy of a network, which is almost solely dependent on the type of activation function used. One common feature of all activation functions is that they are differentiable functions, which allows the Backpropagation (BP) algorithm to be implemented used in the training phase of neural networks.

Activation functions can be either linear or non-linear functions. In mathematics, a linear function is defined as a function that has a linear relationship between the input and output variables. In other words, the rate of change of the input and output is constant. The simplest form of a linear function is $y = mx + n$ where x and y are the independent and dependent variables, respectively and m, n are constants. The degree of the independent variable, x, is always 1 and the graph of a linear function is always a straight line. If a linear function is used as an activation function in a neural network, the network can only adapt to the linear changes of the input, because their boundary is linear. However, since neural networks usually deal with real-world problems which possess ample amounts of non-linear characteristics, they must be able to learn about erroneous data [58]. This can be achieved by using non-linear functions, where the relationship between the input and output variables is not linear and there is at least one curvature when graphed. Thus, non-linear activation functions are generally used instead of linear activation functions in order to add non-linearity to the output of the layers, making the network capable of modelling complex, high-dimensional and non-linear mappings between the input and output. The most commonly used types of activation functions are given below.

2.2.3.1 Sigmoid Activation Function

The sigmoid function, also called the logistic function, is a continuously differentiable, smooth, S-shaped non-linear activation function. The equation of the sigmoid function is given in Equation 2.9 and its graph is given in Figure 4. Sigmoid function converts a vector of real input values to a vector of their normalized values in the inclusive interval [0, 1]. The domain of sigmoid is the set of real numbers and the range is [0, 1]. Similar to the other types of activation functions, the sigmoid function is also differentiable, so the slope of the function can be computed at any given point. The function is monotonic, which refers to any function's strictly non-increasing or non-decreasing nature. In the sigmoid function's case, it is strictly non-decreasing. However, while the sigmoid function is monotonic, its derivative is not. Sigmoid is commonly used in models where the probability of the inputs needs to be predicted, since probability of any event occurring is conveniently between 0 and 1, where 0 means there is no possibility of the event happening and 1 means it will definitely happen. These probabilistic values can be treated as the probabilities of the data points for a particular class, thus sigmoid can intuitively be used in binary classification tasks where the output is two classes. However, due to the vanishing gradient problem, where the gradient converges to zero, sigmoid is usually avoided, because it makes training neural networks difficult.

$$f(x) = \frac{1}{1 + e^{x}} \tag{2.9}$$

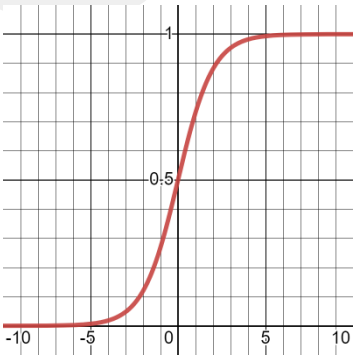


Figure 4: Sigmoid activation function

2.2.3.2 Softmax Activation Function

The softmax function is a more generalized sigmoid activation function, which is used in classification tasks consisting of multiple output classes, unlike the sigmoid function which is typically used for binary classification. Similar to the sigmoid function, softmax takes the input as a vector of real numbers and converts the values to a probability distribution. The output of softmax is also a vector of the same length and each element in the output vector corresponds to the probability of the input belonging to a specific class. Softmax function is also continuously differentiable. The equation of the sigmoid function is given in Equation 2.10.

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \tag{2.10}$$

2.2.3.3 Hyperbolic Tangent Activation Function

Similar to the sigmoid function, the hyperbolic tangent ($tanh$) function is also a continuously differentiable, smooth, S-shaped curve. However, unlike the sigmoid function, it ranges from -1 to 1, inclusive. The advantage $tanh$ has against the sigmoid function is that the negative input values will be mapped strongly negative and the input values that are zero will be mapped in the neighborhood of zero in the graph. $tanh$ is also monotonic, whereas its derivative is not monotonic and is mostly used for classification tasks between two classes. The equation of tanh is given in Equation 2.11 and its graph is given in Figure 5.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1 + e^{-x}} = 2sigmoid(x) - 1 \tag{2.11}$$

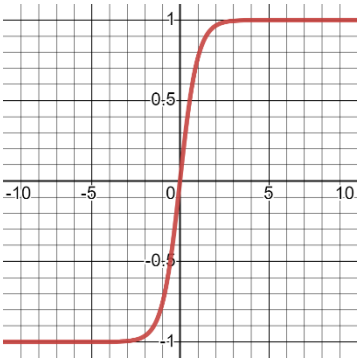


Figure 5: $tanh$ activation function

2.2.3.4 Rectified Linear Unit Activation Function

Rectified Linearu Unit (ReLU) is the default activation function for several types of neural networks, because it makes training easier and improves the performance of the network. If the value of x is negative, $f(x)$ is equal to 0. If the value of x is positive, $f(x)$ is equal to that positive number. In other words, if x is a positive number, the function keeps it as it is, and if it is a negative number, changes it to zero. Thus, its range is the interval $[0, \infty)$. The equation of ReLU is given in Equation 2.12. As given in Figure 6, ReLU is half rectified from the bottom. The ReLU function and its derivative are both monotonic. One disadvantage it has is that since all negative values immediately become 0, the model's ability to fit and train data properly decreases. This is called the dying ReLU problem.

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & x < 0 \end{cases} \tag{2.12}$$

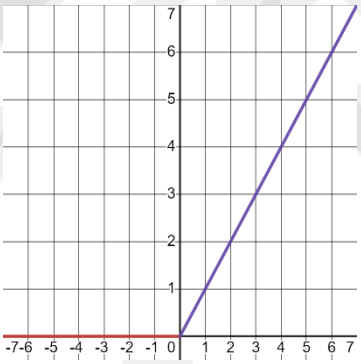


Figure 6: ReLU activation function

2.2.3.5 Leaky ReLU

Leaky ReLU was introduced to overcome the dying ReLU problem. The leak increases the range of the ReLU function. If the value of the input number is positive, the output is that same number. If the input number is negative, the output is the product of a small constant number a and the input itself. The constant number a is usually 0.01. If a is not equal to 0.01, then it is called a Randomized ReLU. This way, the range of Leaky ReLU is $\pm\infty$. Leaky and Randomized ReLU functions are both monotonic, as well as their derivatives. The equation of Leaky ReLU is given in Equation 2.13 and its graph is given in Figure 7.

(2.13)

$$f(x) = \begin{cases} x, & x > 0 \\ ax, & x < 0 \end{cases}$$

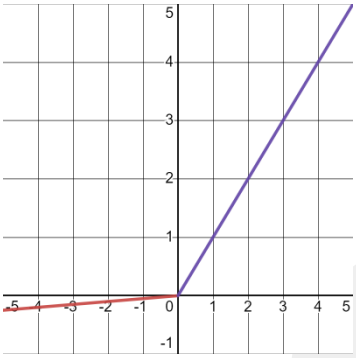


Figure 7: Leaky ReLU activation function

2.2.4 Backpropagation Algorithm

The BP algorithm is an iterative algorithm commonly used for training feedforward ANNs. An input vector representing the states of the input units is fed to the network. States of the input units in this context refers to the activation values of the input units. For example, when working with image data, these input units usually represent the pixel values of the input image. The state of the input units changes iteratively during the forward pass of the network. Forward pass refers to the process of passing the input through the layers of the network where each layer performs a specific operation on the input to transform it in a way that can be passed to the subsequent layer so that lastly, the output layer can generate the desired predictions or decisions based on the transformed input. As the input units are passed through the hidden layers, the states of these units are changed in accordance with Equations 2.14 and 2.15, as presented in the original paper [59]. In Equation 2.14, x_j is a linear function of the outputs, y_j, of the units connected to unit j and of the weights, w_{ji}, on these edges. At this point, a bias term can be added to Equation 2.14, which is a learnable parameter that improves the versatility of the network by allowing it to learn a distinct bias for each neuron. If a bias term is added, Equation 2.14 becomes Equation 2.16.

$$x_j = \sum_i y_i w_{ji} \tag{2.14}$$

$$y_j = \frac{1}{1 + e^{-x_j}} \tag{2.15}$$

$$x_j = \sum_i (y_i w_{ji} + b_i) \tag{2.16}$$

In the case of having a fixed, finite number of input-output cases, the total error, E, can be computed using Equation 2.14, by comparing the actual and desired output vectors for each case [59]. Total error is given in Equation 2.17, where c is an index over input-output cases, j is an index over output units, y is the actual state of an output unit and d is its desired state. In order to optimize the network, the total error must be minimized. The minimization of the total error is achieved using the gradient descent algorithm.

$$E = \frac{1}{2} \sum_c \sum_j \left(y_{j,c} - d_{j,c}\right)^2 \tag{2.17}$$

2.2.4.1 Gradient Descent

Gradient descent is an optimization algorithm commonly used in training neural networks as well as other ML and non-ML models. The goal of neural networks is to update the model parameters with each iteration in order to minimize the loss function, which increases the network's ability to make accurate predictions. Loss function is a mathematical function that measures the difference between the predicted and actual output of a network. Although loss function is similar to the total error, they are not the same, but closely related. The total error is used on the training set to measure the overall performance of the network, whereas the loss function is used to measure the performance of the network on individual training samples. When a neural network is in its training phase, the goal is to minimize the total error by updating the weights and biases of the network using optimization algorithms like gradient descent.

Gradient descent can be applied to all differentiable and convex functions. A function is called differentiable if the derivatives exist for all points in its domain. A function is called a convex function if a line segment can be drawn between any two points, x_1, x_2 in its domain, which does not cross the function at any point, thus staying on the inside of the function. This can be expressed mathematically as Equation 2.18, where λ is the point's location whose value is between 0 and 1. The convex nature of a function can also be determined by computing its second derivative with respect

to the independent variable. If the second derivative is greater than zero for all points in its domain, then the function is called a convex function.

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2) \tag{2.18}$$

Gradient is the slope of a curve at a given point. When the function has only one independent variable, the slope of the function is simply the first derivative of the function at the given point. When the function has two or more independent variables, the slope is a vector of the function's partial derivatives with respect to each independent variable, along each main axis, which is called gradient. The gradient of a function with n independent variables at a given point t is given in the Equation 2.19.

$$\nabla f(t) = \begin{bmatrix} \partial f / \partial x_1 \\ \partial f / \partial x_2 \\ \vdots \\ \partial f / \partial x_n \end{bmatrix} \tag{2.19}$$

Gradient descent optimization algorithm uses the gradient at a particular position to compute the next position. The initial position is iteratively scaled by a hyperparameter called the learning rate and the scaled position is subtracted from the initial position. This step is carried out iteratively. This process can be expressed mathematically as Equation 2.20, where η is the learning rate.

$$p_{n+1} = p_n - \eta \nabla f(p_n) \tag{2.20}$$

2.2.4.2 Minimization of the Total Error

The backward pass, also called backpropagation, process begins with the computation of the partial derivative of E, given in Equation 2.17 with respect to each output unit, $\partial E / \partial y$. The summation operator is a linear operator, which means it can be differentiated term-by-term, where each term is a particular case, c. Differentiating Equation 2.17 with respect to y_j gives:

(2.21)

$$\frac{\partial E}{\partial y_j} = y_j - d_j$$

In order to compute $\partial E/\partial x$, the Leibniz Chain Rule must be applied, which is used when a function y is a differentiable function of a function t and t is a differentiable function of a function x. Applying the Chain Rule to Equation 2.17, gives:

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} . \frac{\partial y_j}{\partial x_j} \tag{2.22}$$

The term dy_j/dx_j can be computed by differentiating Equation 2.15 with respect to x_j:

$$\begin{aligned} y_j &= (1 + e^{-x_j})^{-1} \\ \frac{dy_j}{dx_j} &= -(1 + e^{-x_j})^{-2} . (-e^{-x_j}) \\ &= \frac{-e^{-x_j}}{(1 + e^{-x_j})^2} \\ &= \frac{1/y_j - 1}{\left(1/y_j\right)^2} \\ &= y_j - y_j^2 \\ &= y_j(1 - y_j) \end{aligned} \tag{2.23}$$

Substituting the result obtained from Equation 2.23 into Equation 2.22, we get:

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} . y_j(1 - y_j) \tag{2.24}$$

Taking the partial derivative of the total error with respect to x_j allows us to understand how a change in the total input x to an output unit will affect the total error [59]. Additionally, the total input is a linear function of the states, which is in other

words the activation values, of the previous layers, and the weights of the edges, which is apparent in Equation 2.14. Similarly, we can differentiate the total error with respect to these states, y_j, and weights, w_{ji}, in order to understand how total error is affected with changes to y_j and w_{ji} . Since x_j is a function of y_j and w_{ji}, the Chain Rule must be applied to compute $\partial E/\partial y_i$ and $\partial E/\partial w_{ji}$. Equation 2.25 is the partial derivative for a weight w_{ji} from i to j . Equation 2.26 represents how the effect of i on j changes the total error for the output of the ith unit.

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} . \frac{\partial x_j}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} . y_j \tag{2.25}$$

$$\frac{\partial E}{\partial y_i} = \frac{\partial E}{\partial x_j} . \frac{\partial x_j}{\partial y_i} = \frac{\partial E}{\partial x_j} . w_{ji} = \sum_j \frac{\partial E}{\partial x_j} . w_{ji} \tag{2.26}$$

$\partial E/\partial y$ can be computed for any unit in the penultimate layer if $\partial E/\partial y$ is given for all units in the last layer. All layers in a network can be swept repeating these computations, starting from the penultimate layer to the earlier layers. $\partial E/\partial w_{ji}$ can also be calculated for the weights. This is, as the name suggests, the backward pass process. The weights can either be updated after each input-output case or after all cases are swept using gradient descent by a particular learning rate, as given in Equation 2.20.

In ML models, including CNNs, an epoch, which can be thought of as an iteration, is the operation of sweeping the entire training set to completion during training. In an epoch, all training samples are processed once, predictions are determined and compared to the actual labels and model parameters, which are weights and biases, are updated based on the computed loss value. The aim is to iteratively adjust the model parameters so that the loss function can be minimised and the model performance can be improved. Models typically have multiple epochs for training. The number of epochs is a hyperparameter which is set manually before training, which refers to the number of times the training set will be processed by the model. Determining the number of epochs for a model depends on the size of the dataset and convergence nature of the model as well as the level of complexity of the task-at-hand. It is usually set after a trial-and-error process where the model is run using different

number of epochs and determined when it is observed that the loss does not improve any further despite increasing the number of epochs or when the model's performance seem to worsen. Training the model with too few epochs can result in underfitting whereas training the model with too many epochs can result in overfitting. Underfitting would impair the model's ability to learn embedded patterns present in the data fed to the model. Overfitting would cause the model to learn the training data too well and make it difficult for the model to generalize the learned patterns and fail to perform well on new data.

CHAPTER III
PROPOSED MODELS

DL algorithms, CNNs in particular, have transformed the field of computer vision. As mentioned in Chapter II, there are numerous uses of CNNs, pattern recognition being the most common one. In the recent years, as DL algorithms have been significantly improved, CNNs are regularly being applied in action recognition problems, which is a similar problem to violence recognition. To that end, two 3D CNN models were created in this work for detecting violent behavior in video data. In this chapter, the details of the proposed models are presented.

Two 3D CNN models were developed and proposed in this work. 3D CNNs are usually preferred to 2D CNNs when dealing with video data if we want to use only one type of neural network. The difference of 2D and 3D CNNs was mentioned in Chapter II in detail. In a nutshell, videos are sequences of consecutive images and while images only have spatial information, the difference between the frames must be taken into account when working with video data. The motion information between the frames adds another dimension, *time*, to be dealt with. 2D CNNs can only deal with spatial information, so using a 2D CNN by itself would cause a significant loss in the temporal sense.

Generally, there are two types of model structures developed with CNNs for video classification tasks. The first one is using a 2D CNN as a spatial feature extractor and adding a second deep learning architecture like Recurrent Neural Networks (RNNs) for temporal feature extraction as in [34] [29]. The second one is using a 3D CNN which are capable of dealing with spatio-temporal features without requiring additional ANNs such as RNNs as in [22] [47] or some type of supervised or unsupervised learning algorithms such as SVMs.

3D CNNs can deal with both spatial and temporal features, which results in no loss in motion information. Therefore, 3D CNNs are preferred in this work to avoid the complexity of adding another algorithm to the model. Thus, 3D CNNs were chosen for video classification in the scope of this work.

3.1 MODEL 1: 3D CNN

One of the models proposed in this work is an end-to-end, trainable 3D CNN. This CNN was built using the Sequential class available in Keras API (Application Programming Interface), which groups a linear stack of layers into a Model object. The Sequential class created consists of 2 3D Convolution, 2 3D Max-Pooling and 2 Batch Normalization layers, a Flatten layer and a fully-connected Dense layer. All of these layers were imported from Keras and added to the Sequential model using the *add()* method. This Sequential class created is given in Figure 8.

```
model = tf.keras.Sequential()
model.add(layers.Input(shape=(input_shape[1:])))
model.add(layers.Conv3D(filters=16, kernel_size=(3, 7, 7), padding='valid', activation='relu'))
model.add(layers.MaxPooling3D(pool_size=(2, 2, 2), padding='valid'))
model.add(layers.BatchNormalization())
model.add(layers.Conv3D(filters=16, kernel_size=(3, 7, 7), padding='valid', activation='relu'))
model.add(layers.MaxPooling3D(pool_size=(2, 2, 2), padding='valid'))
model.add(layers.BatchNormalization())
model.add(layers.Flatten())
model.add(layers.Dense(2, activation='softmax'))
```

Figure 8: The Sequential class created for the implementation of the 3D CNN

Since video data (e.g avi files) is used in the experiments, the input data is shaped as a 5-dimensional object with dimensions [batch_size, number_of_frames, height, width, channels]. Number of frames is set as 10, which means 10 frames from each video will be used. Height and width are the dimensions of each frame, which are both set at 224. Channels dimension refers to the colour scheme used, which in this case is RGB. In this colour scheme, all colours are represented as a combination of three primary colors which are red, green, and blue. Batch size is the number of samples used in each iteration in the training phase. The dataset is split into batches of 8 during training and fed to the model to compute gradients of the loss function. The parameters are then updated using the Adam [60] optimizer, which is an optimization algorithm used widely in DL algorithms. Adam optimizer combines the benefits of RMSProp and AdaGrad, which are also optimizers. AdaGrad adapts the learning rate based on the gradients of each parameter and RMSProp scales the learning rate using a moving average of the squared gradient. Learning rate is a hyperparameter used in training that regulates the step size of the weight updation and is usually set before training. A too large learning rate might result in missing a local extrema and cause the model to diverge and a too low learning rate might result in a lengthened training time. After several experiments, 10^{-4} was decided on for the learning rate of the proposed CNN.

There are different types of loss functions, but the function used in the proposed model is Sparse Categorical Cross-Entropy, which is commonly used in multi-class classification problems. There is also a Binary Cross-Entropy, which is used in classification problems with two outcome classes. Although Binary Cross-Entropy could be used in this model, since Sparse Categorical Cross-Entropy works for binary classification as well, it was chosen to allow the network to be generalized and be adapted to multi-class problems as well as binary. The loss function and the optimizer work together to optimize the network, thus improve its performance. In each iteration, loss is calculated and optimized accordingly using the optimizer.

Sparse Categorical Cross-Entropy loss function has the same formula as Categorical Cross Entropy. The difference is that one-hot encoded labels are used in Categorical Cross-Entropy whereas integer encoded labels are used in Sparse Categorical Cross Entropy. The sparse encoding of the labels decreases the memory usage as well as computational resources. For both loss functions, the probabilities of the predicted class are compared with the actual class. In the formula, given in Equation 3.1, where n is the number of classes, t_i is the truth label and p_i is the softmax probability for the ith class, the negative logarithm probability of the correct class label weighted by the true label is calculated. The reason why negative log probability loss functions are commonly used in ML problems is that minimizing negative log of the true class label will reinforce the network to assign higher penalties to incorrect predictions with higher probabilities and lower penalties to incorrect predictions with lower probabilities. The logarithmic nature of the penalties results in large differences being close to 1 and small differences being close to 0.

$$L_{CE} = -\sum_{i=1}^{n} t_i \log p_i$$
$$= -\sum_{i=1}^{n} t_i \log\left(\frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_i}}\right) \quad (3.1)$$

The layout of the proposed model is given in Figure 9. The layout was created using the *plot_model()* method available in Keras Utilities. Parameters in a neural network model are the weights and biases used for computation in all neurons. There are 128,082 total parameters in the model, 128,018 being trainable and 64 being non-

trainable. The summary of the model is given in Figure 10, which was taken during the implementation of the 3D CNN.

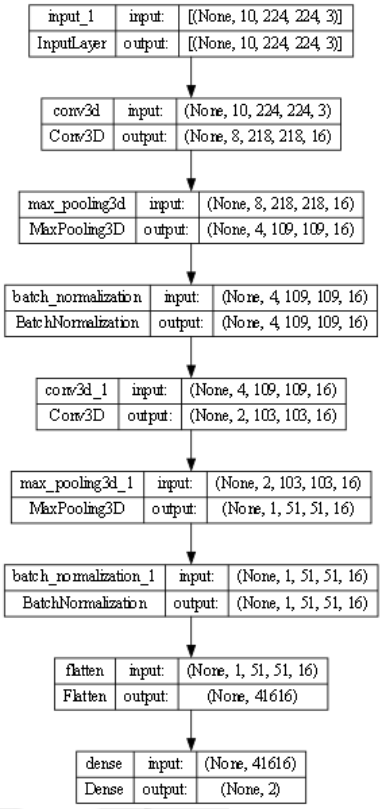


Figure 9: Structural layout of the proposed 3D CNN model

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv3d (Conv3D)             (None, 8, 218, 218, 16)   7072

 max_pooling3d (MaxPooling3D  (None, 4, 109, 109, 16)  0
 )

 batch_normalization (BatchN  (None, 4, 109, 109, 16)  64
 ormalization)

 conv3d_1 (Conv3D)           (None, 2, 103, 103, 16)   37648

 max_pooling3d_1 (MaxPooling  (None, 1, 51, 51, 16)    0
 3D)

 batch_normalization_1 (Batc  (None, 1, 51, 51, 16)    64
 hNormalization)

 flatten (Flatten)           (None, 41616)             0

 dense (Dense)               (None, 2)                 83234

=================================================================
Total params: 128,082
Trainable params: 128,018
Non-trainable params: 64
```

Figure 10: Model summary of the proposed 3D CNN model

After experimenting with several layer hyperparameters for the proposed 3D CNN, the most suitable ones for the task-at-hand were determined. For the 3D convolutional layers, the number of filters is 16, kernel size is (3, 7, 7), padding is valid, stride is 1 and the activation function is ReLU. For the max pooling layers, the kernel size is (2, 2, 2) and padding is valid. Batch Normalization and Flatten layers do not accept any parameters, because they use the output from the preceding layer and apply certain transforms to it before passing it to the next layer. The output of the Dense layer is 2, since the model has two possible outcome classes, fight and non-fight. The activation function used in the Dense layer is the Softmax function, which normalizes the input values into a probability distribution.

The proposed CNN model has significantly fewer parameters than most existing models. The model was trained on a large dataset, which is further explained in Chapter IV. Accuracy to some extent was compromised in order to make the model more efficient yet still effective. The goal is to design the network so that it can be embedded on surveillance systems and even mobile devices. The proposed model can be made to achieve this goal, due to its small size, low power consumption and little memory requirements. In order to use this model in surveillance systems with limited resources, the training of the model can be handled in a separate computerized environment first and be embedded in a surveillance system only for classification. If the training is done successfully on a large dataset, the model will be able to classify new data easily.

3.2 MODEL 2: MOVINET 3D CNN

Transfer learning is used for the second model proposed in this work. In this section, transfer learning and the second proposed model will be defined.

3.2.1 Transfer Learning

Training neural networks is a computationally intense and time-consuming process with high memory requirements. Even when there are no computational or memory-related limitations, there may be a limited amount of data available for the training of a new model. Transfer learning is a useful solution to the problems training neural networks face. Assume that there is a ML model already trained on a large dataset for a task such as image classification for automobiles. Since it has been trained, the model holds the knowledge learned from the dataset to be used for the

classification task-at-hand. Also assume that there is another task, image classification for all vehicles. The knowledge learned from the automobile classification can be used for vehicle classification, which is a similar but different task. This transfer of knowledge is called transfer learning, which is a ML technique that reuses a pre-trained model with learned knowledge to train another model developed for a different but similar problem or domain. The purpose of transfer learning is to decrease the computational intensity and memory requirements, therefore increase the performance and generalization of the developed model.

3.2.2 MoViNets

Mobile Video Networks (MoViNets) [55] are a family of efficient video classification models (3D CNNs) that support online inference, which means running trained AI models to make predictions, on streaming video. It was designed to make 3D CNNs memory and computation-efficient to enable online inference. Three steps were proposed by [55] to achieve this goal. These steps are given below, as described in the original work of [55].

3.2.2.1 MoViNet Search Space

A MoViNet search space was defined in order to allow Neural Architecture Search (NAS) to effectively balance spatio-temporal feature representations [55]. The process of finding the optimal neural network architecture is usually carried out manually using the trial-and-error technique to search for the most suitable model design including layer types and depth, parameters and hyperparameters. NAS is a ML technique that enables the automation of figuring out the optimal neural network architecture for the task-at-hand. NAS methods eliminate the need for the trial-and-error technique and use algorithms to search for the optimal architecture automatically.

Their base search space was built on MobileNetV3[56], an efficient pre-trained CNN model that provides a strong baseline for mobile CPUs [55]. Each 2D block in MobileNetV3 was expanded to handle 3D video input. The dimensions of the input are 50 × 2242 denoted by $T \times S^2$ and the frame stride was set as $\tau = 5$. The search was conducted over the base filter width, $cbase$, for each block. For the additional temporal dimension, the 3D kernel size, denoted by $k^{time} \times (k^{space})^2$ was defined within each layer with the choices (1×3×3), (1×5×5), (1×7×7), (5×1×1), (7×1×1), (3×3×3), (5×3×3). Making these choices enables combining various dimensional

representations, thereby increasing the network's receptive field [55]. No temporal downsampling is applied to enable frame-wise prediction. Squeeze-and-Excitation (SE) blocks were used to capture spatio-temporal information using 3D average pooling. The scaling of the search space was expanded to searching over all scalings of all architectures to find the optimal design, rather than choosing a model and scaling it accordingly.

3.2.2.2 Stream Buffers

The MoViNet search space produced a collection of adaptable 3D CNNs, but their memory requirements increased with the number of input frames, making them impractical for processing long videos. To tackle this issue, the stream buffer mechanism was introduced in order to reduce memory consumption. This mechanism stores feature activations at the endpoints of subclips, enabling the expansion of the temporal receptive field across subclips without requiring any additional computations. The feature map, F_i, of the buffer combined with the subclip along the temporal dimension is computed by Equation 3.2 and the buffer is updated to Equation 3.3 when processing the next clip, where B denotes the buffer, x_i^{clip} denotes the subclip being processed at that moment, $\oplus$ denotes concatenation and $[-b:]$ denotes a selection of the last b frames, which denotes the length of the buffer of the concatenated input.

$$F_i = f(B_i \oplus x_i^{clip}) \tag{3.2}$$

$$B_{i+1} = (B_i \oplus x_i^{clip})_{[-b:]} \tag{3.3}$$

In order to fit 3D CNNs' operations to the stream buffer, all temporal convolutions in 3D CNNs are replaced with Causal Convolutions (CausalConvs) [57], which makes them unidirectional along the temporal dimension. Causality in CNNs refers to the ability of predicting the output of the model using the past and present inputs, not depending on any future inputs. In addition, Cumulative Global Average Pooling (CGAP) is employed to compute global average pooling that encompasses the temporal dimension. This can be calculated as a cumulative sum for any activations up to frame T', which is given in Equation 3.4 where x denotes a tensor of activations. To enable causal calculation of CGAP, a single-frame stream buffer is maintained,

storing the cumulative sum up to T'. Causal SE is presented as a method which is used to multiply the spatial feature map at frame t with the SE computed from $CGAP(x, t)$. Additionally, a sine-based fixed positional encoding (PosEnc) scheme is utilized to directly utilize the frame index as the position. The resulting vector is then summed with the CGAP output before applying the SE projection [55].

$$CGAP(x, T') = \frac{1}{T'} \sum_{t=1}^{T'} x_t \tag{3.4}$$

3.2.2.3 Temporal Ensembles

Although the stream buffer mechanism of MoViNets effectively reduce memory consumption, a little amount of accuracy is lost in the process. To regain this accuracy, an ensembling strategy is employed. This approach involves training two MoViNets individually, where both models have same network design, but operating at half the frame-rate while maintaining the temporal duration. During inference, a video is fed as input to each model, where one of the models have frames offset by one frame. The unweighted logits from both models are then averaged using arithmetic mean before applying softmax [55]. While these models might have lower accuracy separately, this two-model ensemble provides a higher accuracy, while preserving the same Floating Point Operations (FLOPs) as a single model.

In this work, MoViNet-A0 model is used, which was chosen because it is faster to train than the other models the family of MoViNets. The pre-trained MoViNet model was downloaded from the TensorFlow models (tf-models-official) library.

CHAPTER IV
EXPERIMENTS

As mentioned previously, two 3D CNN models were created for this work. In this chapter, the details of their experiments, their results and evaluations are presented, as well as a comparison between the models.

4.1 DATASET DESCRIPTIONS

There are several datasets created specifically for violence detection tasks. The videos in some datasets vary in context, background noise and resolution whereas the videos in some of them have consistency. Nonetheless, none of the datasets consist of high quality videos, which is actually preferred, because low quality videos would result in a more accurate evaluation of the performance of the proposed model. After all, the main goal of building a violence detection model is to equip surveillance systems with the network and most security footage do not have high resolution feed. In contradiction, having a large number of low resolution videos would result in low accuracy percentages. After appropriate research and analysis, 4 datasets were chosen to be used in this work. Each dataset come with unique advantages and disadvantages. Descriptions of the datasets that will be used in this work are given below.

4.1.1 Hockey Fights Dataset

Hockey fights dataset [16] was created by collecting 1000 clips of action from hockey games of the National Hockey League (NHL). Each clip consists of 50 frames of 720x576 pixels labelled as "fight" or "non-fight". All clips have the same background and involve similar human actions where either fights or normal hockey game take place. Background consistency and relatively high resolution videos allow the network to be trained easily. However, learned features would not generalize well to other videos with different contexts and low resolution, which would make the network prone to overfitting.

4.1.2 Movies Dataset

Movies Dataset [16] consists of 200 clips collected from action movies, 100 of which contains a fight scene. The clips consists of 360 x 250 pixels. The videos are taken from a selection of scenes from different movies and some of the videos are of the same scene, with the same context and background. If some of the videos of the same scenes were split into training and test sets, it would most probably cause overfitting. There are also not enough videos for the model to be appropriately trained. In addition, the resolution of the videos is relatively high in comparison with other datasets, which would cause the model to underperform in a surveillance setting.

4.1.3 RWF-2000 Dataset

RWF-2000 Dataset [7] consists of 2000 clips captured by surveillance cameras labelled as "violent" and "non-violent". The clips are collected from YouTube and are of several resolutions. The average length of clips is 5 s. Since the videos are from surveillance footage, they are mostly quite low in resolution and the action occurring is often out of the focus of the camera. Although it consists of a relatively large number of videos, training a network using this dataset is difficult because of the resolution of the videos, which may cause the problem of underfitting.

4.1.4 Real-Life Violence Situations

Real-Life Violence Situations (RLVS) [26] dataset was created in order to overcome the disadvantages of the existing datasets. The RLVS dataset consists of 2000 videos, where 1000 of them are labelled violent and 1000 are labelled non-violent. The clips are of 480p–720p resolutions. Although the video content is richer in context in comparison with the RWF-2000 dataset, there are still a number of videos with out-of-focus action and low resolution.

Existing datasets generally have too much or too little background consistency, too many or too few low resolution videos or consist of only a small number of videos. In order to overcome the limitations of the existing benchmarks, the proposed models in this work are trained and evaluated on a combination of 4 datasets, which are Hockey Fights[16], Movies[16], RWF-2000[7] and Real-Life Violence Situations[26]. The dataset created combining these 4 datasets consist of 5200 videos in total, 2600 of them labelled violent and 2600 labelled non-violent.

In neural networks, videos in the datasets are divided into subsets, which are usually training, test and validation. The training set is used for fitting of parameters based on the observational relationships between the data and their respective labels. The validation set is used for the fitting of hyperparameters and to approximate a model's predictive performance during training [44]. The reason why parameters and hyperparameters are fitted using these separate sets is to avoid overfitting. The test set is not used during training, but it uses the same predictive relationship as the training set and is used for testing.

The majority of the videos in the datasets are used for training. Generally, the split of these subsets are %60, %20, %20 for training, test and validation sets, respectively. The combined dataset used for the experiments of the proposed models consists of 5200 videos where half of them are violent and the other half is non-violent. For the proposed 3D CNN model, 2600 violent and 2600 non-violent videos are each split into 1560 for training, 520 for validation and 520 for test sets. In total, there are 3120 videos for training, 1040 for validation and 1040 for test sets, since the model has two outcome classes. For the proposed MoViNet 3D CNN model, validation set is not needed, because it is a pre-trained network and thus the hyperparameters are already fine-tuned. Therefore, the split is done %60 and %40 between training and test sets, respectively. 2600 violent and 2600 non-violent videos are divided into 1560 for training and 1040 for test sets. In total, there are 3120 videos in the training set and 2080 videos in the test set. In both models, the division of videos was done in random by the algorithm for each experiment. Random selection of videos enable a more precise evaluation of the proposed models.

4.2 EXPERIMENTAL SETUP

All experiments were conducted on the computer 11th Gen Intel(R) Core(TM) i7 64GB, 16 cores with NVIDIA GeForce GTX 1660 SUPER GPU. The codes were written using Python 3.7, the Tensorflow framework and the layers of the network were imported from the Keras library. The outline of the codes were inspired by the publicly-available Tensorflow tutorials [45] and [46], but changed and adapted in accordance with the proposed model.

4.2.1 Libraries

A library in Python is a collection of codes serving a particular purpose. In order to run the proposed neural network models in Python, the necessary libraries were installed and imported. *Remotezip* was used to inspect the contents of the ZIP file in which the combined dataset resided. *Tqdm* was used for the progress bar which showed as the files were split into their subsets. *OpenCV* was used in video processing, performing operations such as extracting frames from the videos. *Einops* was used to simplify the complex tensor operations performed. *Random* was used to shuffle the dataset randomly while splitting the dataset into subsets. *Pathlib* provided a more appropriate file system path representation. *Itertools* provided several functions built for iterators to produce more complex iterators. *Collections* implements container datatypes in addition to Python's built-in containers such as list, set, tuple, etc. *NumPy* is a commonly used open source Python library when working with numerical data. It contains multidimensional array and matrix data structures alongside a large variety of high-level mathematical functions that enables operating on the arrays and matrices. For example, extracted frames from the videos in this work were contained in a *NumPy* array. *Pandas* is an open source Python library that contains DataFrame object for data manipulation, tools for reading and writing data, data alignment and handling missing data, reshaping and merging datasets, etc. *Matplotlib* is used for creating static, animated and interactive visualizations including graphs and model layouts. *Seaborn* is also a data visualization library based on *matplotlib*. It provides a high-level interface for drawing statistical graphics. *Matplotlib* and *Seaborn* were used in this work to create the graphs and confusion matrices. *Shutil* library provides high-level operations on files, which was used in this work for checking whether a file already existed before creating it. Keras is a deep learning API written in Python, running on top of the ML platform TensorFlow used for building ML models and their implementations. It has a variety of applications such as importing pre-trained models, creating existing or custom ML models from scratch and their layers. Keras can be run on CPUs, GPUs and TPUs (TensorFlow Processing Unit).

4.3 EVALUATION METRICS

The goal of the proposed models is to determine whether an input video contains a fight or not. After the training phase, videos in the test set are fed to the network and classified as fights and non-fights. In this phase, the network predicts the

label of the input videos based on the learned features. Since both models use supervised learning, the samples are already labelled as fight or non-fight. To that end, there are four possible outcomes of the classification executed by the models. A video of a fight can either be classified correctly as a fight or incorrectly as a non-fight. Similarly, a non-fight video can either be classified correctly as a non-fight or incorrectly as a fight. These predicted values can be found in the confusion matrices created for the test set for each experiment, and certain evaluation metrics can be calculated using these values to evaluate the performance of the models.

In the extent of the experiments conducted for this work, the event of detecting a fight has a truth value of 1 (positive) and not detecting a fight has a truth value of 0 (negative). In the confusion matrices created in this work, the vertical axis represents the actual action, which is the actual label of a video, and the horizontal axis represents the predicted action, which is the label of a video predicted by the network. The terms True and False refer to the correct or incorrect predictions made by the network, respectively, whereas Positive and Negative refer to the labels, which in this case are Positive for fights and Negative for non-fights. If a fight is predicted as a fight, it is called a true positive (TP) and if a fight is predicted as a non-fight, it is called a false negative (FN). If a non-fight is predicted as a fight, it is called a false positive (FP) and if a non-fight is classified as a non-fight, it is called a true negative (TN). Their corresponding elements in the confusion matrices are top left corner for TP, top right corner for FN, bottom left corner for FP and bottom right corner for TN. The evaluation metrics calculated based on these values are accuracy, precision, recall and F1-score.

4.3.1 Accuracy

Accuracy is calculated by dividing the number of correctly classified videos, which means the prediction matches the actual label, by the total number of videos. The accuracy metric does not make the distinction of whether a fight is detected or not, it is simply the measure of the correctly predictions made by the network, which correspond to the TP and TN elements in the confusion matrix. The formula for calculating accuracy is:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

4.3.2 Recall

Recall is calculated by dividing TP values by the total number of positive samples in the dataset. In the case of the proposed models, this is equal to dividing the number of fights classified correctly as fights, by the total number of fight videos. Recall is the measure of the model's ability to correctly predict the positive samples. If the number of FN cases are high, the recall of the model would be a small. Recall is calculated with the formula:

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

4.3.3 Precision

Precision is calculated by dividing TP values by the total number of positive predictions made by the model. In the case of the proposed models, this is equal to dividing the number of fights classified correctly as fights, by the total number of videos classified as fights. As the name suggests, precision is the measure of how precise a model is. If the number of FP cases are high, which are non-fight videos classified incorrectly as fight videos, the value of precision would decrease. Precision is calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

4.3.4 F1-Score

F1-score is the harmonic mean of precision and recall values of a model. It is a combined measure of precision and recall, in the range of [0, 1]. If the value of the F1-score is close to 1, it means the model has high precision and recall. In most cases, the values of precision and recall are close to each other, so F1-score is usually a consistent evaluation metric. It is calculated as follows:

$$F1 - Score = \frac{2 \times precision \times recall}{precision + recall} \tag{4.4}$$

4.4 RESULTS AND MODEL EVALUATIONS

In this section, separate evaluations of both models based on the experiments are presented. For each calculation, 6-digit rounding is used.

4.4.1 3D CNN Model

The proposed 3D CNN model was experimented with using different numbers of epochs and it was determined that the most suitable number was 100 epochs. Any number of epochs less than 100 resulted in low accuracy. It was observed that loss ceased to improve when the number of epochs was increased beyond 100. The average length of an epoch was 640 seconds, so the average run-time of the model including both training and inference phases was 17.8 hours with the GPU used in the experiments. Although this is a considerably long time, it is common for the training of 3D CNNs to take this long, especially considering the size of the dataset the proposed model is being run on and that GPU used in the experiments has fairly low computational power. The run-time is highly dependent on these two factors. The combined dataset has a large number of videos and most of the videos are taken from surveillance cameras which are of low resolutions with diverse contexts. Computing the feature maps for these types of videos is difficult for the model and thus, it takes longer than it would for high resolution videos with similar contexts. However, changing the GPU with one that has higher computational power would result in shortening the run-time of the model. Regardless, a run-time of 17.8 hours for a 3D CNN where videos are being processed can be considered as satisfactory performance. The proposed 3D CNN model is a simple 3D CNN which lacks complex layers with too many floating-point and matrix operations. As a result, the model has 128,082 total parameters. This number is significantly lower than most proposed 3D CNNs. In a way, some accuracy was compromised in order to make the model smaller in size with lower computational requirements.

The average accuracy of the proposed 3D CNN model was calculated as the average of all 5 experiments, which was 82.46%. Considering the size of the model and low resolutions of the videos in the dataset, this result is favorably decent. The accuracy, loss, recall, precision and F1-scores were computed for each experiment, which are presented in Table 1. The reason why these values vary significantly for each experiment is that the dataset was randomly shuffled and split into training, test and validation tests before each training. Accuracy-loss graphs were drawn during the

implementation of the model as well as confusion matrices for both training and test sets. These figures for each experiment are given in Appendix 1. As mentioned above, accuracy and loss values were computed during implementation whereas recall, precision and F1-score values were calculated using their respective formulas using the values in the confusion matrix of the test set.

Table 1: Accuracy, loss, recall, precision and F1-score percentages of each experiment for the proposed 3D CNN model

Experiment	Accuracy	Loss	Recall	Precision	F1-Score
1	0.8144	1.5946	0.813462	0.790654	0.801896
2	0.8250	1.4465	0.844231	0.809963	0.809963
3	0.8231	1.4109	0.803846	0.832669	0.818004
4	0.8163	1.8286	0.875000	0.789931	0.830292
5	0.8442	1.3360	0.871154	0.834254	0.852305

4.4.2 MoViNet 3D CNN

Transfer learning was leveraged in the proposed MoViNet 3D CNN model. The model was pre-trained on the action dataset Kinetics 600 [62], which has 600 classes of action, each with at least 600 video clips. The proposed model only looked for similar patterns in the new data, which significantly decreased the number of epochs and the run-time of the model. After experimenting with higher and lower number of epochs, it was observed that the value of loss stopped improving after 25 epochs. Thus, each experiment involving the proposed model was executed with 25 epochs. The average length of each epoch was 625 seconds, which resulted in the run-time of the model being an average of 4.3 hours. Since the model was only required to become familiar with the new data it was fed instead of being trained from scratch, this run-time is unusually short for a 3D CNN, yet expected since it leveraged transfer learning.

The average accuracy of the model was calculated as 91.712%, which is the average of the 5 experiments that were conducted. This result is remarkable for only 25 epochs, which is an indication of the effectiveness and efficiency of transfer learning. Since the MoViNet model was trained on an action dataset, it allowed for the proposed model to recognize violent activity even though most of the combined dataset used in this work consisted of surveillance footage with low resolution videos and out-of-focus action. The accuracy, loss, recall, precision and F1-scores were computed for each experiment, which are presented in Table 2. Since the dataset was shuffled

randomly at the beginning of each experiment, these values vary by some amount. The graphs for accuracy-loss drawn during implementation and confusion matrices of test sets for each experiment are given in Appendix 2. Evaluation metrics are calculated using their respective formulas using the values in the confusion matrix created for the test set.

Table 2: Accuracy, loss, recall, precision and F1-score percentages of each experiment for the proposed MoViNet 3D CNN model

Experiment	Accuracy	Loss	Recall	Precision	F1-Score
1	0.9149	0.3796	0.902885	0.914314	0.908564
2	0.9178	0.3895	0.925000	0.908404	0.916627
3	0.9245	0.3741	0.926923	0.924257	0.925588
4	0.9087	0.5206	0.857692	0.951974	0.902377
5	0.9197	0.3721	0.920192	0.921965	0.921078

4.5 COMPARISON OF THE PROPOSED MODELS

The proposed 3D CNN model was trained from scratch in each experiment with no weight initialization whereas the MoViNet 3D CNN model uses transfer learning with a pre-trained model. It was anticipated that the MoViNet model would outperform the 3D CNN model in all aspects. 3D CNN model learned the data fed to it with no prior knowledge and the weights were initialized at 0 as default. The weights were updated in each epoch based on the optimizer's computations against the value produced by the loss function. When the loss function stopped improving around the hundredth epoch, it was determined that the model had reached its highest performance. Reaching this performance required 100 epochs which took around 17.8 hours. Considering that the model had no weight initialization, had no prior learned information and no other complex machine learning algorithms were added to it, the average accuracy reached by the model was considerably high. It must be noted that this accuracy was reached with an optimal number of parameters, 128,082, which is significantly lower than most proposed violence detection models.

On the other hand, the proposed MoViNet model outperformed the 3D CNN, with a significant accuracy difference, which is around 10%. This accuracy difference was achieved with one-forth number of epochs of the 3D CNN as well as run-time, which was around 4.3 hours. This run-time is significantly shorter than other proposed violence detection models. The model was not trained from scratch, rather only became familiar with the new data it was fed. Another disadvantage of 3D CNNs is

that they do not support online inference due to their high computational resources and memory requirements. Thus, 3D CNNs are not suitable to work on mobile devices unless they are improved to do so. However, MoViNet models are more computationally efficient compared to 3D CNNs and they have significantly lower memory usage. This allows them to work on mobile devices. Interpreting the results, it is clear that transfer learning has a significant advantage against other simple DL models. Considering all of these factors contributing to the performance of both models, they still achieved impressive results within the range of their respective expectations. The accuracy, loss, recall, precision and F1-score values for both models, calculated as the average of their 5 respective experiments are given in Table 3.

Table 3: Overall accuracy, loss, recall, precision and F1-score percentages for both models, calculated by taking the average of each evaluation metric for all experiments of each model

Model	Accuracy	Loss	Recall	Precision	F1-Score
3D CNN	82.46%	1.523320	84.15386%	81.14942%	82.58478%
MoViNet 3D CNN	91.712%	0.407180	90.65384%	92.41828%	91.48468%

Since the dataset was randomly shuffled and split into training, test and validation sets for the 3D CNN model and only training and test sets for the MoViNet 3D CNN model at the beginning of each experiment, the evaluation metrics given in Table 1 and Table 2 vary by some amount. In order to interpret the results comprehensively, the standard deviation for each metric of both models was calculated using Equation 4.5, where n is the number of data points, x_i is the values of each data and x is the average (mean) of $\bar{x}$.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})}{n-1}} \tag{4.5}$$

Since standard deviation is a measure of the amount of variation of a set of values, it can be said that a low standard deviation value means minimal variation occurred in the set of values. Judging from the standard deviation values given in Table 4, shuffling the dataset at the beginning of each experiment did not cause a large amount of dispersion in the results, but did allow for a more accurate evaluation of the proposed models. The standard deviation values for the evaluation metrics are mostly consistent with each other. It also must be noted that the overall standard deviation

values for the MoViNet model is around 10 times lower than the 3D CNN model. This shows that shuffling the dataset randomly had less effect on the MoViNet model where transfer learning was leveraged than it had on the 3D CNN model. This difference reflects on the consistency achieved by transfer learning.

Table 4: Standard deviation values of accuracy, loss, recall, precision and F1-score for both models

Model	Accuracy	Loss	Recall	Precision	F1-Score
3D CNN	0.011827	0.194875	0.032454	0.216102	0.018410
MoViNet 3D CNN	0.005861	0.063762	0.028905	0.016760	0.009385

4.6 COMPARISON WITH OTHER WORKS IN LITERATURE

In this work, a combined dataset is used to evaluate the proposed the models. There are no other works in literature that uses the same combined dataset, so a thorough comparison of the models with other works could not be achieved. However, in order to compare the proposed models with other works in literature, a total of 8 separate experiments were conducted using the datasets in the combined dataset individually. Both proposed models were trained with the individual datasets once. The results of these experiments are presented in Table 5 and Table 6 for the 3D CNN model and the MoViNet 3D CNN model, respectively.

Table 5: Accuracy, loss, recall, precision and F1-score values of each experiment for the proposed 3D CNN model

Dataset	Accuracy	Loss	Recall	Precision	F1-Score
Hockey Fights	0.9400	0.3307	0.9400	0.9400	0.9400
Movies	1.0000	4.8123×10^{-4}	1.0000	1.0000	1.0000
RWF-2000	0.7125	1.8605	0.8150	0.687764	0.745996
RLVS	0.8950	0.7723	0.8450	0.944134	0.891821

Table 6: Accuracy, loss, recall, precision and F1-score values of each experiment for the proposed MoViNet 3D CNN model

Dataset	Accuracy	Loss	Recall	Precision	F1-Score
Hockey Fights	0.9200	0.4118	0.8950	0.942105	0.917949
Movies	1.0000	0.0083	1.0000	1.0000	1.0000
RWF-2000	0.8788	0.4966	0.9150	0.853147	0.882992
RLVS	0.9638	0.2072	0.9550	0.984536	0.969543

Interpreting the results from the individual dataset experiments, it can be seen that both models reached a 100% accuracy with the Movies dataset, which is a result that has been achieved by most works in literature. The MoViNet model significantly

outperformed the 3D CNN model with the RWF-2000 and Real Life Violence Situations datasets. On the other hand, the 3D CNN model slightly outperformed the MoViNet model in the Hockey Fights dataset. It should be noted that the MoViNet experiments were conducted with 25 epochs whereas the 3D CNN experiments were conducted with 100 epochs for the sake of not changing the experimental setting. The Hockey Fights dataset is rather an easy dataset to train and considering that the 3D CNN model ran with 100 epochs, it is probable that it outperformed the MoViNet model with 25 epochs. If the number of epochs of the MoViNet model was to be increased, it would most likely outperform the 3D CNN model eventually. The Real Life Violence Situations and RWF-2000 datasets are more difficult to train, so the accuracy gap between the models were expected.

It is noteworthy to mention that the run-time of the 3D CNN experiments were 1.25 hours, 16.7 minutes, 11.1 hours and 5.5 hours for the Hockey Fights, Movies, RWF-2000 and Real Life Violence Situations datasets, respectively. The run-time of the MoViNet experiments, on the other hand, are 17.7 minutes, 5 minutes, 2.7 hours and 1.3 hours for the Hockey Fights, Movies, RWF-2000 and Real Life Violence Situations datasets, respectively. The MoViNet model reached mostly higher accuracy values in a significantly shorter amount of time with individual dataset experiments, which was also the case with the experiments using the combined dataset. While the proposed 3D CNN model reached high accuracy values with the individual dataset experiments, the run-time of the model is too long. Thus, the results from the individual experiments still point to the efficiency and effectiveness achieved by using the MoViNet model.

High accuracy values have been reached by other proposed models in literature for the Hockey Fights and Movies datasets. Both proposed models have achieved a 100% accuracy with the Movies dataset. Although the accuracy of the Hockey Fights experiments of both models could not come close to other proposed models, the MoViNet model presented in this work performed extremely well with the RWF-2000 and Real Life Violence Situations datasets. The MoViNet model came close to the accuracy values reached by other works in literature with the RWF-2000 dataset. In addition, the MoViNet model especially outperformed the original paper where the Real Life Violence Situations dataset was introduced, by a significant accuracy difference of approximately 25%. The models and accuracy values of the experiements carried out for individual datasets in some other proposed works in literature are given

in Table 7. Overall, both proposed models performed well with the combined dataset as well as individual datasets.

Table 7: Accuracy percentages of some other works in literature

Model	Year	Hockey Fights	Movies	RWF-2000	RLVS
MoSIFt+HIK[16]	2011	90.9%	89.5%	-	-
C3D[25]	2019	96.0%	99.9%	-	-
VGG16+LSTM[26]	2019	95.1%	99.0%	-	71.5%
C3D+SVM[48]	2020	98.5%	-	-	-
VGG16+ConvLSTM[33]	2021	99.1%	100%	92.4%	-
Xception+LSTM[29]	2021	96.5%	98.3%	-	-
SepConvLSTM-M[32]	2021	99.5%	100%	89.75%	-
CNN+BiLSTM[35]	2022	94.9%	92.9%	-	-

CHAPTER V
CONCLUSION

The aim of this thesis was to extensively study CNNs, give mathematical explanations to how they work and implement and compare two types of 3D CNN models to efficiently and effectively detect violence on video data. One of the models was a simple 3D CNN whereas the other model was a 3D CNN which leveraged transfer learning. A combination of 4 different publicly-available violence detection datasets was used in the experiments with a total of 5200 videos, half of which are violent and the other half are non-violent. After 5 experiments were conducted on both models using this combined dataset, it was quantitatively determined that the transfer learning model, MoViNet, resulted in more accurate results in a significantly shorter amount of time. The performance of the MoViNet model being higher than the simple 3D CNN model was reasonably expected, because MoViNet model was pre-trained on an action dataset and the model only needed to become familiar with the new data whereas the simple 3D CNN model was trained from scratch. Architecture designs, underlying theoretical foundations and implementation specifics of both models as well as their comparisons were explained in detail.

In future works, the simple 3D CNN model is planned to be improved by adding new layers to make the model more efficient such as attention layers and use weight initialization to shorten the training-time and to increase accuracy. This model can also be combined with classifiers such as SVMs other DL models such as RNNs. Additionally, the 3D CNN model can be made smaller in size and lighter in computational intensity, eventually making them suitable for mobile device inference. The MoViNet model, on the other hand, has reached its anticipated potential efficiency with the data it was fed and the GPU it was implemented on. It can be further improved by using a larger dataset for the training and a more powerful GPU.

REFERENCES

[1] PERŠ Janez, SULIĆ Vildana, KRISTAN Matej, PERŠ Matej, POLANEC Klemen and KOVAČIČ Stanislav (2010), "Histograms of optical flow for efficient representation of body motion", *Pattern Recognition Letters*, vol. 31, issue 11, pp. 1369–1376.

[2] CAO Yongqiang, CHEN Yang and KOSHLA Deepak (2015), "Spiking deep convolutional neural networks for energy-efficient object recognition", *International Journal of Computer Vision*, vol. 113, issue 1, pp. 54–66.

[3] IBM (2022), *Convolutional Neural Networks*, https://www.ibm.com/topics/convolutional-neural-networks, DoA. 13.10.2022.

[4] FISHER Robert (2007), *CAVIAR Test Case Scenarios*, https://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/, DoA. 12.10.2022.

[5] SULTANI Waqas, CHEN Chen and SHAH Mubarak (2019), *UCF-Crime*, https://paperswithcode.com/dataset/ucf-crime, DoA. 12.10.2022.

[6] ZHANG Tao, YANG Zhijie, JIA Wenjing, YANG Baoqing, YANG Jie and HE Xiangjian (2016), "A new method for violence detection in surveillance scenes", *Multimedia Tools and Applications*, vol. 75, issue 12, pp. 7327–7349.

[7] CHENG Ming, CAI Kunjing and LI Ming (2021), "RWF-2000: an open large scale video database for violence detection", *25th International Conference on Pattern Recognition (ICPR)*, pp. 4183–4190, Milan, Italy.

[8] CHEN Ming-Yu and HAUPTMANN Alexander (2009), "Mosift: Recognizing human actions in surveillance videos", *In Technical Report, CMU-CS-09-161*, Pittsburgh, USA.

[9] HE Kaiming, ZHANG Xiangyu, REN Shaoqing and SUN Jian (2016), "Deep residual learning for image recognition", *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, Las Vegas, Nevada, USA.

[10] LAPTEV Ivan (2005), "On space-time interest points", *International journal of computer vision*, vol. 64, issue 2, pp. 107–123.

[11] SIMONYAN Karen and ZISSERMAN Andrew (2014), "Very deep convolutional networks for large-scale image recognition", *arXiv preprint*, arXiv:1409.1556, pp.1-14.

[12] SOOMRO Khurram, ZAMIR Amir Roshan and SHAH Mubarak (2012), "UCF101: A dataset of 101 human actions classes from videos in the wild", *arXiv preprint*, arXiv:1212.0402, pp. 1-7.

[13] SANDLER Mark, HOWARD Andrew, ZHU Menglong, ZHMOGINOV Andrey and CHEN Liang-Chieh (2018), "Mobilenetv2: Inverted residuals and linear bottlenecks", *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, Salt Lake City, Utah, USA.

[14] CHOLLET François (2017), "Xception: Deep learning with depthwise separable convolutions", *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1251–1258, Honolulu, Hawaii, USA.

[15] KRIZHEVSKY Alex, SUTSKEVER Ilya and HINTON Geoffrey E. (2017) "Imagenet classification with deep convolutional neural networks", *Communications of the ACM*, vol. 60, issue 6, pp. 84–90.

[16] BERMEJO NIEVAS Enrique, DENIZ SUARES Oscar, BUENO GARCIA Gloria and SUKTHANKAR Rahul (2011), "Violence detection in video using computer vision techniques", *International conference on Computer analysis of images and patterns*, pp. 332–339, Seville, Spain.

[17] ARCEDA V. Machaca, FABIÁN K. Fernández and GUTIERREZ, Juan Carlos (2016), "Real time violence detection in video", *International Conference on Pattern Recognition Systems*, pp. 6-7, Talca, Chile.

[18] HASSNER Tal, ITCHER Yossi and KLIPER-GROSS, Orit (2012), "Violent flows: Real-time detection of violent crowd behavior", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–6, Providence, Rhode Island, USA.

[19] XU Long, GONG Chen, YANG Jie WU Qiang and YAO Lixiu ((2014), "Violent video detection based on MoSIFT feature and sparse coding", *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3538–3542, Florence, Italy.

[20] XIE Jianbin, YAN Wei, MU Chundi, LIU Tong, LI Peiqin and YAN Shuicheng (2016), "Recognizing violent activity without decoding video streams", *Optik*, vol. 127, issue 2, pp. 795–801.

[21] ARCEDA V. Machaca, FABIÁN K. Fernández, LAURA P.C. Laguna, TIJO J.J. Rivera and CÁCERES J.C. Gutiérrez (2016), "Fast face detection in violent video scenes", *Electronic Notes in Theoretical Computer Science*, vol. 329, pp. 5–26.

[22] TRAN Du, BOURDEV Lubomir, FERGUS Rob, TORRESANI Lorenzo and PALURI Manohar (2015), "Learning spatiotemporal features with 3d convolutional networks", *Proceedings of the IEEE international conference on computer vision*, pp. 4489–4497, Washington, DC, USA.

[23] SUDHAKARAN Swathikiran and LANZ Oswald (2017), "Learning to detect violent videos using convolutional long short-term memory", *14th IEEE international conference on advanced video and signal based surveillance (AVSS)*, pp. 1–6, Lecce, Italy.

[24] ABDALI Al-Maamoon R. and AL-TUMA Rana F. (2019), "Robust real-time violence detection in video using cnn and lstm", *2nd Scientific Conference of Computer Sciences (SCCS)*, pp. 104–108, Baghdad, Iraq.

[25] ULLAH Fath U. Min, ULLAH Amin, MUHAMMAD Khan, HAQ Ijaz Ul and BAIK Sung Wook (2019), "Violence detection using spatiotemporal features with 3D convolutional neural network", *Sensors*, vol. 19, issue 11, pp. 2472.

[26] SOLIMAN Mohamed Mostafa, KAMAL Mohamed Hussein, NASHED Mina Abd El-Massih, MOSTAFA Youssed Mohamed, CHAWKY Bassel Safwat and KHATTAB Dina (2019), "Violence recognition from videos using deep learning techniques", *Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pp. 80-85, Cairo, Egypt.

[27] SUMON Shakil Ahmed, GONI Raihan, HASHEM Niyaz Bin, SHAHRIA Tanzil and RAHMAN Rashedur M. (2020), "Violence detection by pretrained modules with different deep learning approaches", *Vietnam Journal of Computer Science*, vol. 7, issue 01, pp. 19–40.

[28] LIANG Jinhua, ZHANG Tao and FENG Guoqing (2020), "Channel compression: Rethinking information redundancy among channels in CNN architecture", *IEEE Access*, vol. 8, pp. 147265–147274.

[29] SHARMA Sarthak, SUDHARSAN B., NARAHARISETTI Saamaja, TREHAN Vimarsh and JAYAVEL Kayalvizhi (2021), "A fully integrated violence detection system using CNN and LSTM", *International Journal of Electrical & Computer Engineering (2088-8708)*, vol. 11, issue 4, pp. 3374-3380.

[30] KANG Min-seok, PARK Rae-Hong and PARK Hyung-Min (2021), "Efficient spatiotemporal modeling methods for real-time violence recognition", *IEEE Access*, vol. 9, pp. 76270–76285.

[31] ALDAHOUL Nouar, KARIM Hezerul Abdul, DATTA Rishav, GUPTA Shreyash, AGRAWAL Kashish and ALBUNNI Ahmad (2021), "Convolutional Neural Network-Long Short Term Memory based IOT Node for Violence Detection", *IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, pp. 1–6, Kota Kinabalu, Sabah, Malaysia.

[32] ISLAM Zahidul, RUKONUZZAMAN Mohammad, AHMED Raiyan, KABIR Md. Hasanul and FARAZI Moshiur (2021), "Efficient two-stream network for violence detection using separable convolutional lstm", *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, Shenzhen, China.

[33] MUGUNGA Israel, DONG Junyu, RIGALL Eric, GUO Shaoxiang, MADESSA Amanuel Hirpa and NAWAZ Hafiza Sadia (2021), "A Frame-Based Feature Model for Violence Detection from Surveillance Cameras Using ConvLSTM Network", *6th International Conference on Image, Vision and Computing (ICIVC)*, pp. 55–60, Qingdao, China.

[34] VOSTA Soheil and YOW Kin-Choong (2022), "A cnn-rnn combined structure for real-world violence detection in surveillance cameras", *Applied Sciences*, vol. 12, issue 3, pp. 1021.

[35] TALHA Khalid Raihan, BANDAPADYA Koushik and KHAN Mohammad Monirujjaman (2022), "Violence Detection Using Computer Vision Approaches", *IEEE World AI IoT Congress (AIIoT)*, pp. 544–550, Seattle, USA.

[36] VO-LE Cuong, VO Hung Sy, VU Thien Duy and SON Nguyen Hong (2022), "Violence Detection using Feature Fusion of Optical Flow and 3D CNN on AICS Violence Dataset", *IEEE Ninth International Conference on Communications and Electronics (ICCE)*, pp. 395–399, Seoul, South Korea.

[37] O'SHEA Keiron and NASH Ryan (2015), "An introduction to convolutional neural networks", *arXiv preprint*, arXiv:1511.08458, pp.1-11.

[38] NAM Jeho, ALGHONIEMY Masoud and TEWFIK Ahmed H. (1998), "Audiovisual content-based violent scene characterization", *Proceedings International Conference on Image Processing. ICIP98 (Cat. No. 98CB36269)*, vol. 1, pp. 353–357, Chicago, Illinois, USA.

[39] SIVIC Josef and ZISSERMAN Andrew (2003), "Video Google: A text retrieval approach to object matching in videos", *IEEE International Conference on Computer Vision*, vol. 3, pp. 1470–1470, Nice, France.

[40] CHENG Wen-Huang, CHU Wei-Ta and WU Ja-Ling (2003), "Semantic context detection based on hierarchical audio models", *Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pp. 109–115, Berkeley, California, USA.

[41] CLARIN Christine T., DIONISIO Judith Ann M., ECHAVEZ Michael T. and NAVAL Prospero C. (2005), "DOVE: Detection of movie violence using motion intensity analysis on skin and blood", *PCSC*, vol. 6, pp. 150–156.

[42] CHEN Datong, WACTLAR Howard, CHEN Ming-Yu, GAO Can, BHARUCHA Ashok and HAUPTMANN Alex (2008), "Recognition of aggressive human behavior using binary local motion descriptors", *30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 5238–5241, Vancouver, British Columbia, Canada.

[43] YUJIAN Li and BO Liu (2007), "A Normalized Levenshtein Distance Metric", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, issue 6, pp. 1091-1095.

[44] SCHILLING Fabian (2016), *The effect of batch normalization on deep convolutional neural networks* (Master's Thesis), KTH, Stockholm.

[45] Tensorflow (2022), *Video classification with a 3D convolutional neural network*, https://www.tensorflow.org/tutorials/video/video_classification, DoA. 06.20.2022.

[46] Tensorflow (2023), *Transfer learning for video classification with MoViNet*, https://www.tensorflow.org/tutorials/video/video_classification, DoA. 01.10.2023.

[47] SONG Wei, ZHANG Dongliang, ZHAO Xiaobing, YU Jing, ZHENG Rui and WANG Antai (2019), "A novel violent video detection scheme based on modified 3D convolutional neural networks", *IEEE Access*, vol. 7, pp. 39172–39179.

[48] ACCATOLI Simone, SERNANI Paolo, FALCIONELLI Nicola, MEKURIA Dagmawi Neway and DRAGONI Aldo Franco (2020), "Violence detection in videos by combining 3D convolutional neural networks and support vector machines", *Applied Artificial Intelligence*, vol. 34, issue 4, pp. 329–344.

[49] JI Shuiwang, XU Wei, YANG Ming and YU Kai (2012), "3D convolutional neural networks for human action recognition", *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, issue 1, pp. 221–231.

[50] IOFFE Sergey and SZEGEDY Christian (2015), "Batch normalization: Accelerating deep network training by reducing internal covariate shift", *International conference on machine learning*, pp. 448–456, Lille, France.

[51] LI Chengyang, ZHU Liping, ZHU Dandan, CHEN Jiale, PAN Zhanghui, LI Xue and WANG Bing (2018), "End-to-end multiplayer violence detection based on deep 3D CNN", *Proceedings VII international conference on network, communication and computing*, pp. 227–230, Taipei City, Taiwan.

[52] VIJEIKIS Romas, RAUDONIS Vidas and DERVINIS Gintaras (2022), "Efficient violence detection in surveillance", *Sensors*, vol. 22, issue 6, pp. 2216.

[53] HUSSAIN Tariq, IQBAL Arshad, YANG Bailin and HUSSAIN Altaf (2022), "Real time violence detection in surveillance videos using Convolutional Neural Networks", *Multimedia Tools and Applications*, vol. 81, issue 26, pp. 38151–38173.

[54] LECUN Yann, JACKEL Lawrence D., BOTTOU Léon, CORTES Corinna, DENKER John S., DRUCKER Harris, GUYON Isabelle, MULLER Urs A., SACKINGER Eduard, SIMARD Patrice and VAPNIK Vladimir (1995), "Learning algorithms for classification: A comparison on handwritten digit recognition", *Neural networks: the statistical mechanics perspective*, vol. 261, issue 276, pp. 2.

[55] KONDRATYUK Dan, YUAN Liangzhe, LI Yandong, ZHANG Li, TAN Mingxing, BROWN Matthew and GONG Boqing (2021), "Movinets: Mobile video networks for efficient video recognition", *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16020–16030, Montreal, Canada.

[56] HOWARD Andrew, SANDLER Mark, CHU Grace, CHEN Liang-Chieh, CHEN Bo, TAN Mingxing, WANG Weijun, ZHU Yukun, PANG Ruoming, VASUDEVAN Vijay, LE Quoc V., ADAM Hartwig (2019), "Searching for mobilenetv3", *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1314–1324, Seoul, Korea.

[57] OORD Aaron van den, DIELEMAN Sander, ZEN Heiga, SIMONYAN Karen, VINYALS Oriol, GRAVES Alex, KALCHBRENNER Nal, SENIOR Andrew and KAVUKCUOGLU Koray (2016), "Wavenet: A generative model for raw audio", *arXiv preprint*, arXiv:1609.03499, pp. 1-15.

[58] SHARMA Sagar, SHARMA Simone and ATHAIYA Anidhya (2017), "Activation functions in neural networks", *Towards Data Science*, vol. 6, issue 12, pp. 310–316.

[59] RUMELHART David E., HINTON Geoffrey E. and WILLIAMS Ronald J. (1986), "Learning representations by back-propagating errors", *Nature*, vol. 323, issue 6088, pp. 533–536.

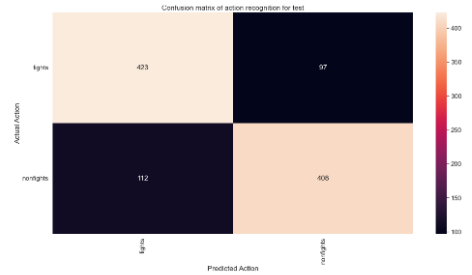
[60] KINGMA Diederik P. and BA Jimmy (2014), "Adam: A method for stochastic optimization", *arXiv preprint*, arXiv:1412.6980, pp. 1-15.

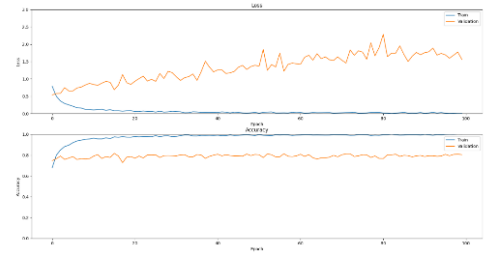
[61] KARPATHY Andrej, TODORICI George, SHETTY Sanketh, LEUNG Thomas, SUKTHANKAR Rahul and FEI-FEI Li (2014), "Large-scale video classification with convolutional neural networks", *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, Columbus, Ohio, USA.

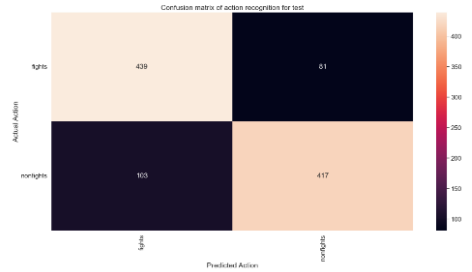
[62] CARREIRA Joao, NOLAND Eric, BANKI-HORVATH Andras, HILLIER Chloe and ZISSERMAN Andrew (2018), "A short note about kinetics-600", *arXiv preprint*, arXiv: 1808.01340, pp. 1-6.

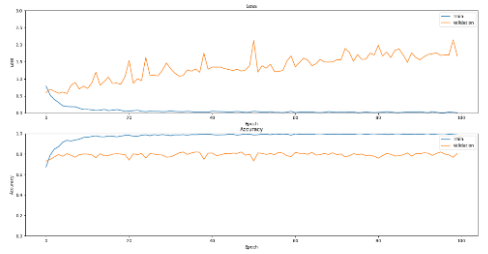
APPENDICES

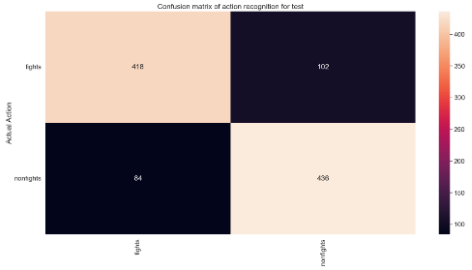
APPENDIX 1: 3D CNN Loss-Accuracy Graphs and Confusion Matrices for the Test Set for Each Experiment

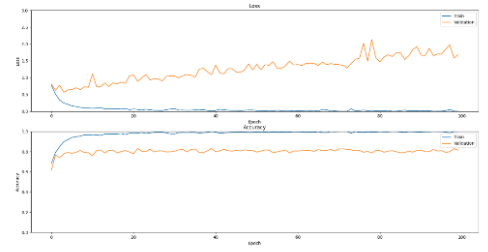


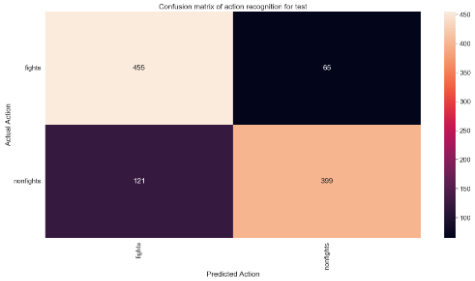


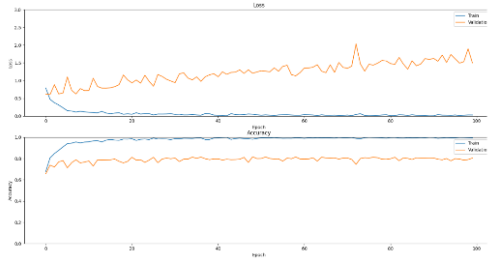


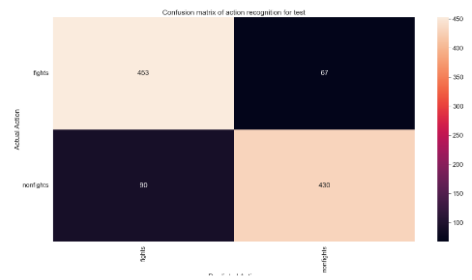


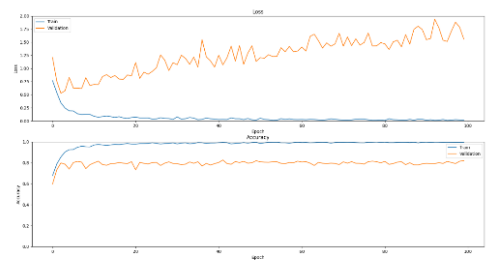












APPENDIX 2: MoViNet 3D CNN Loss-Accuracy Graphs and Confusion Matrices for the Test Set for Each Experiment