Automatic Violence Detection Using Deep Learning Techniques

[21MCA4208]

Submitted By: -

MITHUN R [ENG22MCA046]
CHETHAN SD [ENG22MCA035]
ARUN KUMAR A [ENG22MCA026]
AADITYA SREEHARI [ENG22MCA023]

In partial fulfillment for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

DEPARTMENT OF COMPUTER APPLICATIONS

DAYANANDA SAGAR UNIVERSITY

SCHOOL OF ENGINEERING

KUDLUGATE, HOSUR ROAD, BANGALORE -560068

MARCH 2024

ACKNOWLEDGEMENT

We wish to express my salutations to the beloved and highly esteemed institute **DAYANANDA SAGAR UNIVERSITY.** we thank **Dr. Uday Kumar Reedy**,Dean, School of Engineering, Dayananda Sagar University, Bangalore, for his valuable suggestions and expert advice. we would like to thank **Dr. VasanthiKumari P**, Chairman, Department of Computer Applications, Dayananda Sagar University for her valuable guidelines and constant support throughout my project.

We are grateful to our guide and would like to express special thanks to **Prof. Sindhu S**, Assistant Professor, Department of Computer Applications, DSU for her constant support, necessary advice, and valuable guidance that helped me in completing the project/report work successfully.

We would like to thank all the Teaching and non-teaching staff of the Departmentof Computer Applications, DSU for their help. We would like to thank **My parents** and **friends** who have always supported me in every path of the Reportwork.

We perceive this opportunity as a milestone in my career development. We will strive to use the gained skills and knowledge in the best possible way and We willcontinue to work on my improvement in order to achieve the desired career objectives. We hope to continue the cooperation with all of you in the future.

MITHUN R [ENG22MCA046]
CHETHAN SD [ENG22MCA035]
ARUN KUMAR A [ENG22MCA026]
AADITYA SREEHARI [ENG22MCA023]

DAYANANDA SAGAR UNIVERSITY

Kudlu Gate, Hosur Road, Bangalore-560068

DEPARTMENT OF COMPUTER APPLICATIONS

Bonified Certificate

This is to certify that the project titled "Automatic Violence Detection Using Deep Learning Techniques" is a Bonified record of the work done by **MITHUN R [ENG22MCA046], CHETHAN SD [ENG22MCA035], ARUN KUMAR A [ENG22MCA026] AND AADITYA SREEHATRI [ENG22MCA023].**

In partial fulfillment of the requirements for the award of the degree of **Master of Computer Applications** in **DAYANANDA SAGAR UNIVERSITY, BANGALORE**, during the year 2023-2024

Guide	**Chairman**
Prof. Sindhu S	**Dr. Vasanthi Kumari. P**
Assistant Professor	Professor &Chairperson
Dept. of Computer Applications,	Dept. of Computer Applications,
Dayananda Sagar University	Dayananda Sagar University

Project Viva Held on: -

Internal Examiner	**External Examiner**

ABSTRACT

The detection of a violent event in surveillance systems plays a significant role in law enforcement and city safety. However, this leads to a huge pressure on security attendants who need to monitor footage constantly and on a daily basis. In the event of a violent crime, the police department needs to analyze vast amounts of footage quickly in order to obtain any information. As a result, automatically detecting violent incidents from surveillance video data is critical.

The purpose of our project is to develop a monitoring system to detect the presence of violence and non-violence in the video provided by the surveillance cameras, using Deep Learning techniques.

We concentrate on the difficult task of detecting violent activities in video data through surveillance cameras. The overall effectiveness of the mode is measured in terms of accuracy.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	
	LIST OF FIGURES	
1	**INTRODUCTION**	1
	1.1 INTRODUCTION TO THE PROJECT	1
	1.2 RNN	1
	1.3 CNN	2
	1.4 LSTM	3
2	**LITERATURE SURVEY**	5
3	**SYSTEM CONFIGURATION**	7
	3.1 HARDWARE SPECIFICATION	7
	3.2 SOFTWARE SPECIFICATION	7
4	**DATASET USED**	8
5	**DATA PRE-PROCESSING**	9
6	**LIBRARY USED**	10
	6.1 OpenCV	10
	6.2 Keras	10
	6.3 Tensor Flow	10
7	**HYPERPARAMETERS AND METRICS**	11
	7.1 Model accuracy	11
	7.1.2 Loss Function	11
	7.2 HYPERPARAMETERS	12

CHAPTER NO	TITLE	PAGE NO
	7.2.1 Strides	12
	7.2.2 Padding	12
	7.3 Kernel size	14
	7.4 Activation Functions	1
	7.5 Optimizers	1
8	**MODELS IMPLEMENTED**	2
	8.1 ConvLSTM	3
	8.2 VGG16	5
	8.3 Pre-Trained VGG16 with RNN	7
	8.4 Pre-Trained VGG16 with Attention layer	7
	8.5 GRU	7
	8.6 Pre-Trained VGG16 with Dense layer	9
	8.7 Pre-Trained EfficientNet with RNN	10
9	**CONCLUSION**	10
	FUTURE SCOPE	10
	REFERENCES	11
	APPENDICES	11
	A. SOURCE CODE	11
	B. SCREENSHOT	12

LIST OF FIGURES

FIGURE NO	NAME OF THE FIGURE	PAGE NO
1.2.1	RNN NETWORK STRUCTURE	
1.3.1	CNN NETWORK STRUCTURE	
1.4.1	LSTM STRUCTURE	1
1.4.2	EQUATION FOR THE FORWARD PASS	1
2.1	THE PROPOSED MODEL	1
2.2	PROPOSED MODEL ACCURACY	2
7.1.1	EQUATION TO CALCULATE THE ACCURACY	3
7.1.2	EQUATION OF CROSS FUNCTION	5
7.2.1	STRIDES OF IMAGE	7
7.2.2	ZERO PADDING	7
7.4.1	ACTIVATION FUNCTION	7
7.5.1	EQUATION FOR OPTIMIZATION IN ADAM	8
8.1.1	ConvLSTM UNIT	9
8.1.2	EQUATION OF ConvLSTM	10
8.1.3	MODEL SUMMARY OF ConvLSTM	10
8.2.1	NETWORK STRUCTURE OF VGG16	10
8.2.2	MODEL SUMMARY OF VGG16	10
8.3.1	MODEL SUMMARY OF PRE-TRAINED VGG16 WITH DENSE LAYER	11
8.4.1	MODEL SUMMARY OF PRE-TRAINED VGG16 WITH ATTENTION LAYER	11
8.5.1	GRU UNIT	11
8.5.2	MODEL SUMMARY OF GRU	12

FIGURE NO	NAME OF THE FIGURE	PAGE NO
8.6.1	MODEL SUMMARY OF PRE-TRAINED WITH GRU WITH DENSE LAYER	
8.7.1	STRUCTURE OF EFFICIENTNETB6	
8.7.2	MODEL SUMMARY OF PRE-TRAINED EFFICIENT NET	1
9.1	TRANSFORMER UNIT	1

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION OT THE PROJECT

Deep Learning has pushed the limits of what was possible in the Digital Image Process domain. That is not to say that traditional computer vision techniques, which had been steadily improving for years before the rise of Deep Learning, have become obsolete.

In the field of digital image processing, deep learning is used to solve difficult problems (e.g. image colorization, classification, segmentation, and detection). Machine learning includes a subset called deep learning. Artificial Neural Networks, a computing paradigm inspired by the functioning of the human brain, are at the heart of Deep Learning. Deep Learning also increases the usage of computational power and the results observed were efficient, in a considerably less amount of time.

The purpose of this work is to examine different models which detect violence from surveillance cameras and alert the necessary authorities. The CLI Tool helps in an easier user interface, with the input being a video that needs to be classified as violent or non-violent. Since the footage is taken in the form of a video,i.e a path to the video file stored in the system. We use the violence detection system to predict on this video footage and determine whether the source is violent or not. Once the source is determined to be violent the violence detection system alerts the authorities to take action accordingly. The common terminologies used are as follows:

1.2 RNN

Recurrent Neural Networks are one of the well-known and widely implemented Networks in Deep Learning Techniques. Based on Feed Forward Neural Networks, RNNs can process inputs with variable lengths with the help of their internal state. In order to build a temporal sequence, they form connections among the nodes. This allows the network to be a dynamic temporal network. Since we require extracting spatial features too, RNNs can’t be much of a help in detecting the required spatial features.

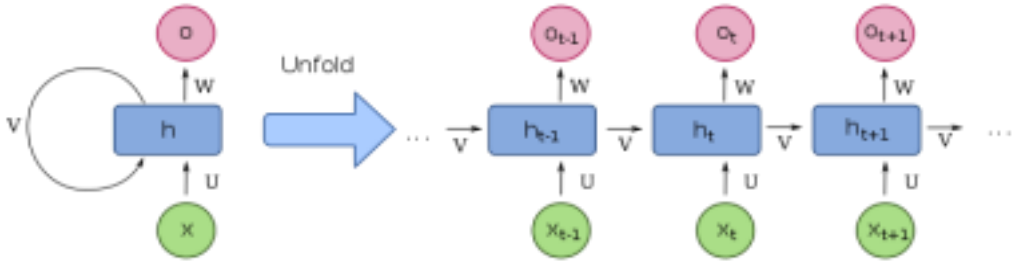


Figure 1.2.1:RNN Network Structure

1.3 CNN

Unlike traditional neural networks, convolutional neural networks employ convolution at least in one layer. These CNNs typically contain an input along with multiple hidden and a final output layer. It is mostly implemented when the input is a type of image. Normally, a CNN's inputs take the form of Tensors as follows: (input width) x (input channels) x (input height) x (number of inputs).

Convolution layers convolve inputs from the previous layers and pass them to the next layer. The fact that Convolution is Shift/Space invariant determines that CNNs are Shift/Space invariant Neural Networks. Once the inputs were passed through the convolution layers the convolved shape turns out to be (feature map width) x (feature map channels) x (feature map height) x (number of inputs).

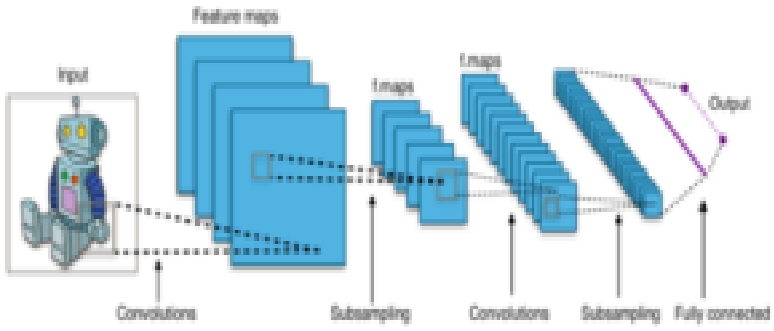


Figure 1.3.1:CNN Network Structure

1.4 LSTM

LSTM is a RNN with gates implemented along with feedback connections . A Typical LSTM contains an input, a cell, an output and finally a forget gate. While training a traditional

RNN using back propagation the long-term gradient can vanish, this problem is also well known as the vanishing gradient problem. Implementing the LSTM can partially solve this problem. Whereas in case of exploding gradient the LSTM might still suffers is renowned for classifying and making predictions based on time series data.

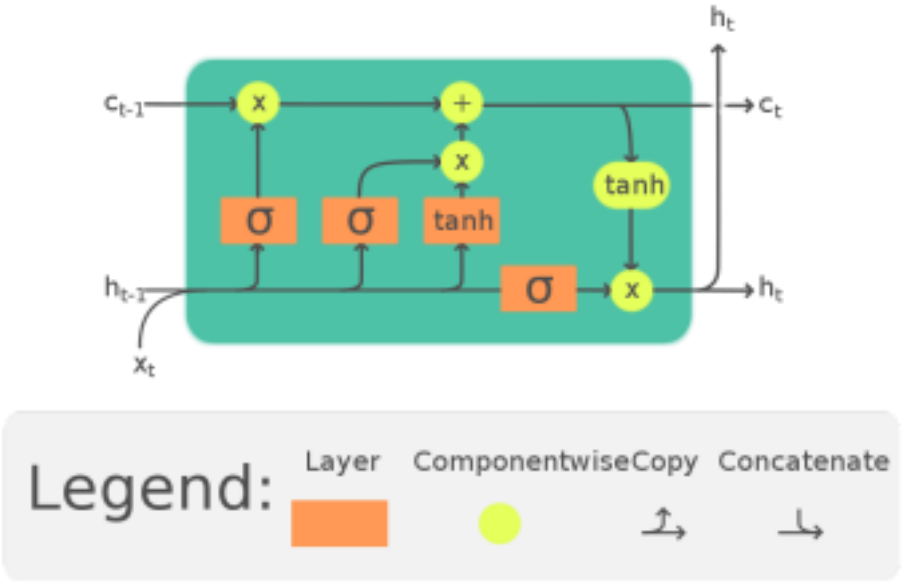


Figure 1.4.1:LSTM Structure

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Figure 1.4.2: Equations for the forward pass of an LSTM unit with a forget gate

CHAPTER 2

LITERATURE SURVEY

In ‘Learning to detect violent videos using convolutional long short-term memory’ [1] by SwathiKiran Sudhakaran and Oswald Lanz, a model using convLSTM is developed for detecting fight scenes. The frames from the videos are resized to 256x256 resolution and 224x224 resolution for training and testing purposes respectively.A comparison of the fully-connected LSTM and convLSTM is also made, with the results revealing that the convLSTM model can create an improved video representation than LSTM with lesser parameters, and also by preventing overfitting. The proposed model using convLSTM is as follows:

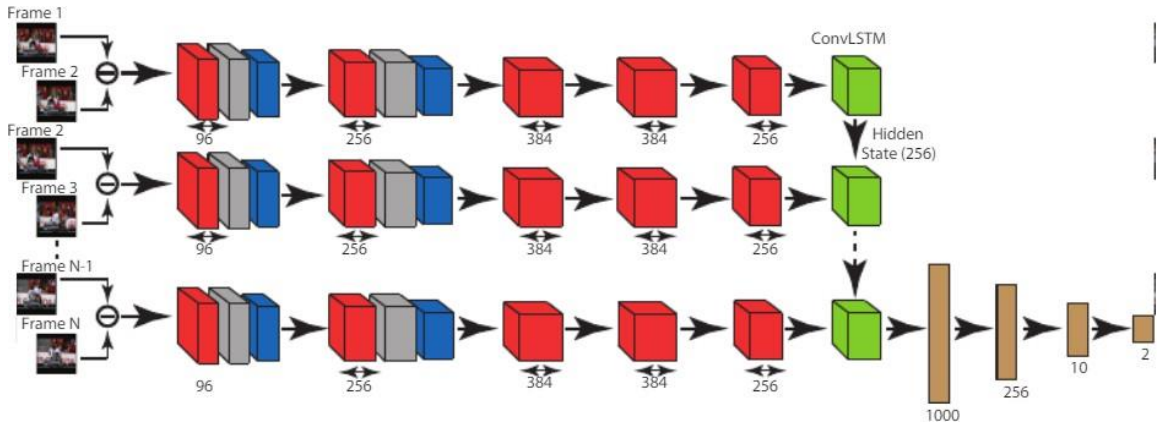


Figure 2.1: The proposed model of ConvLSTM

The frames in question are applied to the model in order. First the convolutional (red) layer, followed by normalisation (grey) layer and the last pooling (blue) layers alternate, with convolutional layers in the end. For classification, the ConvLSTM's hidden state (green) at the final time step is utilised. In convLSTM, 256 filters of size 3x3 and stride 1 were used in all gates. Layers that are fully connected are shown in brown. After each of the convolutional and fully-connected layers, the activation function ReLU is applied. The model was trained on 3 different datasets and the comparison between convLSTM and LSTM based on accuracies is shown as follows:

Model	Accuracy	No. of Parameters
convLSTM	97.1±0.55%	9.6M(9619544)
LSTM	94.6±1.19%	77.5M(77520072)

Figure 2.2: Proposed model, accuracy and number of parameters involved

ConvLSTM has an advantage in terms of accuracy as well as in the number of parameters over LSTM. ConvLSTM is better as it encodes Spatio-temporal features, where the spatial features are encoded by CNN and temporal features with the help of an RNN. Whereas in LSTM, only temporal features are detected as it is only an RNN. ConvLSTM is an appropriate option for fight sequences because it involves looking at both the location of humans and how their motion changes over time. ConvLSTM produces a more accurate representation of the video under consideration. A few terminologies are explained as follows:

In "Temporal Robust Features for Violence Detection"[6], the proposed method in the literature is a Bag of Visual Words Framework containing three different layers.

The first level is made of a Spatio-temporal video descriptor which extracts features from the video that encapsulate the frame pixel values variation according to both spatial configuration and their disposition along the video timeline. The authors have employed the use of Temporal Robust Features for its significantly lesser computational cost.Coding and pooling are the 2 methods in the half- level pipeline. Initially the coding step will quantize the low-level descriptors as per codebook.Whereas the pooling step will summarize by combining the codes into a single feature vector. Fisher Vectors will carry out the extraction.

The third layer deals with training a linear Support Vector Machine for classifying the mid level feature vectors. The usage of TRoF achieved a 72% classification accuracy

CHAPTER 3

SYSTEM CONFIGURATION

3.1 Hardware Specification

- Processor : Core i5 – 5th Gen
- RAM : 16GB
- System type : 64Bit Operating System
- Monitor : Dell 24 inch
- Keyboard : USB keyboard
- Mouse : USB Optical mouse

3.2 Software Specification

- Platform : Microsoft Windows 10
- Designing language : Python 3.9, OpenCV, keras, Tensor flow

CHAPTER 4

DATASETS USED

Since there aren't many open source datasets available to be used specifically for training the system on detecting violence using surveillance cameras, we have concluded that dataset namely "A dataset for automatic violence detection in videos" by Miriana Bianculli, Nicola Falcionelli,Paolo Sernani,Selene Tomassini,Paolo Contardo,Mara Lombardi and Aldo FrancoDragoni [2] as the primary dataset on which model is to be trained.

The Dataset consists of a total of 350 videos, out of which 230 are violent videos and the rest 120 are non-violent videos, the resolution of each video is 1920 x 1080 Pixels .

A different dataset namely "Violence Recognition from Videos using Deep Learning Techniques" by M. Soliman, M. Kamal, M. Nashed, Y. Mostafa, B. Chawky, D. Khattab [3] is considered to check the generalization nature of the models.

There are 2000 videos, for violent and non-violent. 1000 videos each for violent and non-violent classes. A few videos are used for cross referencing the accuracy of the model prediction.

CHAPTER 5

DATA PRE-PROCESSING

The raw video is first converted to a list of frames. The number of frames that were passed to the numpy array as an element from each video is 40. As mentioned above the dataset used is considerably huge and passing each raw video consumes a lot of resources and time. Due to a limitation on the resources, we have chosen to reduce the dimensions of the initial dataset (1920 X 1080 pixels) and reduce each frame to a dimension of 108 x 108 pixels. The frames are converted to a numpy array. A corresponding numpy array containing the labels of the video is also generated, for the model to learn from this data. Declaring a function for extracting the frames from each video and calling it while parsing through each video file will reduce the computation time. This labeled data is then divided into train and test data. 80% of the data belongs to training data and 20% as the data on which model can be cross validated.

CHAPTER 6

LIBRARIES USED

6.1 OpenCV

OpenCV, or Open Source Computer Vision Library, is a collection of functions primarily centred on real-time computer vision. It was initially written in C++ and now most of its libraries are binded with other programming languages like Python and Java. OpenCV is mainly focused on Image Processing in real time. It also supports hardware acceleration i.e usage of GPU.

The functions which are being used in this work are

- cv2.VideoCapture(): VideoCapture uses the path to the video file as an argument.
- cv2.VideoCapture.read(): Read converts the video to multiple images depending on the number of frames that were declared.
- cv2.resize(): Resize deals with resizing the frames depending on the values of height and width declared.

6.2 Keras

Keras is the high-level API of TensorFlow 2, a python API mainly focusing on Deep learning that is placed on top of the TensorFlow module. Keras is simple, flexible, and really powerful. The core data structures of Keras are layers and models, which are really helpful while building and training different models. For more complex architectures such as VGG16 etc Keras functional API can be used.

For the model to be declared and able to deal with sequential data we use the Keras function Model = Sequential, which will be imported from the keras module by 'from keras.models import Sequential'.

The different layers of the model such as max pooling, dropout, attention etc.. can be done by calling the Keras based python functions.

6.3 TensorFlow

TensorFlow is an end-to-end open source library having a vast amount of tools that help in building efficient models focusing on machine learning . It also provides interconnectivity between other python modules such as Keras and PyTorch whose main focus is also on dl.

CHAPTER 7

HYPERPARAMETERS AND METRICS USED FOR MODEL COMPARISON

7.1 Model Accuracy

Accuracy is the main metric that describes how well trained a model is. It describes how well the model is classifying the given input data into the correct classes as declared earlier,i.e it describes how well the model performs across all the classes. Accuracy means Classification/Categorization accuracy, which is nothing but the ratio of the number of accurate predictions to the overall number of samples in the input. In Keras module it is described as Metrics = 'Accuracy'.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

Figure 7.1.1 :Equation to calculate accuracy of a given model

7.1.2 Loss Function

It is a specific method that evaluates how well the model understands the given data and predicts accordingly. The deviation that occurs while prediction from what the actual results should be,is considered to be the loss. If the deviation is considered to be high,the loss function will give out a huge value. Loss functions can be described into two main classes.

Regression Losses and Classification Losses.Classification losses are the one to be chosen for image processing based models,since the model classifies the given input based on the classes that were declared earlier before training the model. Categorical cross entropy is the loss function which is typically used for image type data while train.

$$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

Figure 7.1.2: Equation of Cross Categorical Entropy Loss

7.2 HyperParameters:

7.2.1 Strides

It is one of the components of neural networks(CNN) which deals with the compression of image

data. The number of pixel shifts on the input matrix is known as a stride. The size of the stride filter affects the overall encoded output volume, it is recommended to take an integer as the filter size rather than having a floating point value.

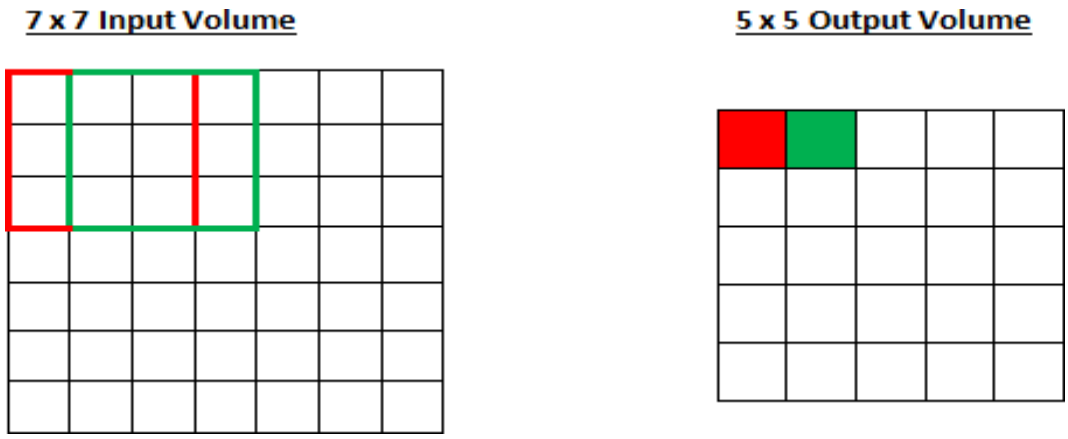


Figure 7.2.1:Strides on an image

7.2.2 Padding

Padding is another important filter of neural networks (CNN), which describes the number of pixels that are being added to an image while the kernel is processing through it. This prevents the image from losing its shape as the image is padded with extra 0's from all sides.

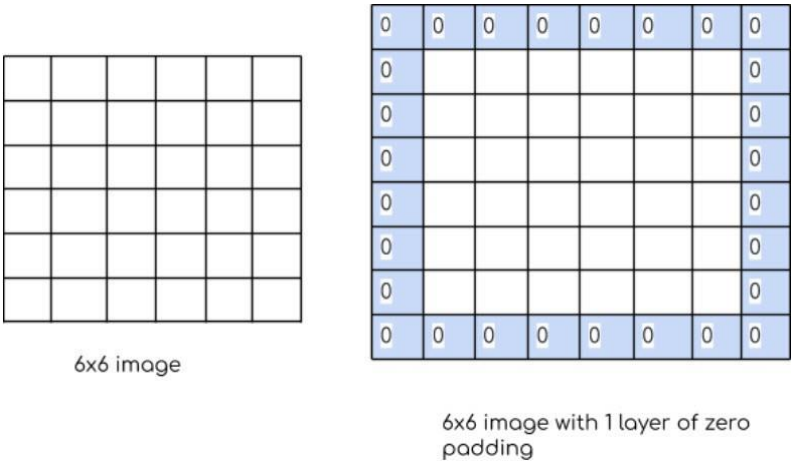


Figure 7.2.2: Applying a zero padding to a 6x6 image

7.3 Kernel size

The kernel size refers to the magnitude of the filter that is used to convolve the feature map, whereas the step size refers to the quantity by which the filter slides.

7.4 Activation Functions

An activation function helps to explain how the weighted sum of inputs is transformed to outputs by nodes of the network in a Neural Network. Typically, Neural Networks contain multiple layers, called input, output, and hidden. There is a single activation function for all hidden layers, whereas the output layer uses a diverse activation function when compared to the hidden layers.

In hidden layers, a nonlinear activation function is used. Some of the nonlinear activation functions like 'ReLu, Sigmoid, and Tanh'. For a CNN we use ReLu as a hidden layer and in RNN Sigmoid and Tanh functions are used as activations in the hidden layer. Output layers in a CNN typically use a softmax activation function, which outputs a vector of values that sum to 1 and helps in classification.

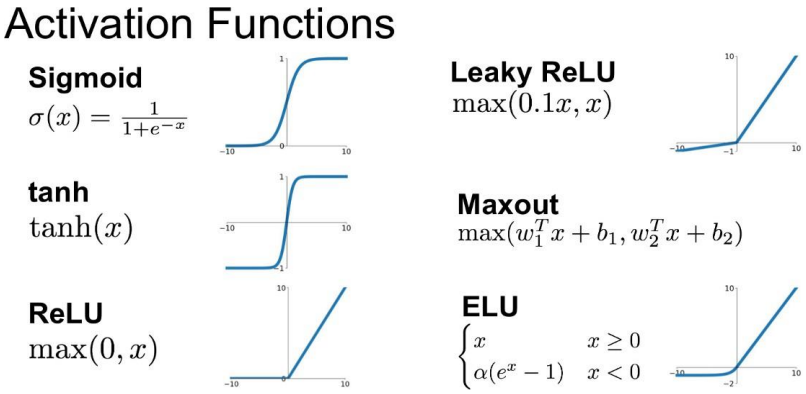


Figure 7.4.1:Different types of Activation Functions

7.5 Optimizers

As the name indicates, optimizers are algorithms that are implemented while training the model in order to increase the model accuracy, learning rate and optimize the losses that are occurring. Optimizers change the weights accordingly to get the best efficiency. There are several optimizers in deep learning such as gradient descent,momentum gradient descent, Adagrad, etc...

Algorithm 1: Adam Optimizer

Initialize $\theta_0, m_0 \leftarrow 0\ , v_0 \leftarrow 0, t \leftarrow 0$

While θ_t not converged

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$

$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

Bias Correction

$\widehat{m_t} \leftarrow \frac{m_t}{1-\beta_1^t}, \widehat{v_t} \leftarrow \frac{v_t}{1-\beta_2^t}$

Update

$\theta_t \leftarrow \prod_{\mathcal{F}, \sqrt{\widehat{v_t}}} \left(\theta_{t-1} - \frac{\alpha \widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon}\right)$

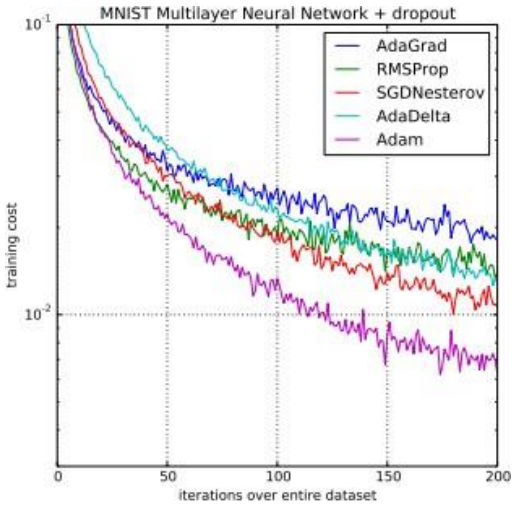


Figure 7.5.1 : Equation for optimization in ADAM and Plot of Performance different optimizers in a multilayer Neural Network

The number of parameters that are being trained is an important aspect of Deep Learning techniques. The shape that is being passed on to the model to train must be equivalently the same and distributed according to the number of parameters that are distributed accordingly to each layer in the respective model.

CHAPTER 8

MODELS IMPLEMENTED

A wide range of Deep Learning Models were trained and compared on the dataset. These Models vary in the approaches taken towards classification.

Below mentioned models are considered to be pre-eminent compared to others:

8.1 ConvLSTM with dense and dropout layers

8.2 Implementing VGG16 Architecture

8.3 Pre-trained VGG16 with RNN

8.4 Pre-trained VGG 16 with Attention Layer

8.5 Pure GRU

8.6 Pre-trained VGG16 with GRU with RNN

8.7 Pre-Trained EfficientNet with RNN

8.1 ConvLSTM

Beginning with the traditional models for classification, Long Short-Term Memory and Convolutional Neural Network, ConvLSTMis a hybrid of CNN and LSTM, where the fully connected layers in LSTM are replaced with CNN. ConvLSTM is a type of RNN (Recurrent Neural Network) with convolutional structures that is used for Spatio-temporal. The ConvLSTM uses the input state and previous state of its nearest neighbours to predict future state of a cell in the grid.

The model contains a 2D ConvLSTM layer consisting of 72 filters and a kernel of a 3 x 3 matrix. After the dropout layer, a Dense layer with 256 neurons with each having ReLU as an activation function follows it. Then another dropout layer is added, which helps avoid overfittin

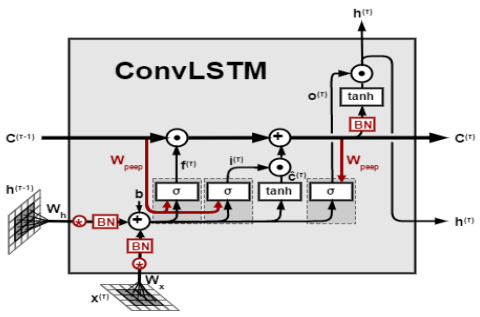


Figure 8.1.1 : ConvLSTM unit

ConvLSTM

$$
\begin{aligned}
i_t &= \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i) \\
f_t &= \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f) \\
\mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c) \\
o_t &= \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o) \\
\mathcal{H}_t &= o_t \circ \tanh(\mathcal{C}_t)
\end{aligned}
$$

Figure 8.1.2: Equation of peephole ConvLSTM

The model summary is as follows:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_lst_m2d (ConvLSTM2D)    (None, 106, 106, 72)      194688
_________________________________________________________________
dropout (Dropout)            (None, 106, 106, 72)      0
_________________________________________________________________
flatten (Flatten)            (None, 808992)            0
_________________________________________________________________
dense (Dense)                (None, 256)               207102208
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 514
=================================================================
Total params: 207,297,410
Trainable params: 207,297,410
Non-trainable params: 0
_________________________________________________________________
```

Figure 8.1.3: Model Summary ConvLST

8.2 VGG16

"VGG" is an acronym for Visual Geometry Group and is a CNN model. K. Simonyan and A. Zisserman introduced their findings in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”[4].

VGG is used to research how the depth of convolution layers is affecting the accuracy of the model while classifying /recognising large scale images .

The Architecture of VGG16 is as follows:

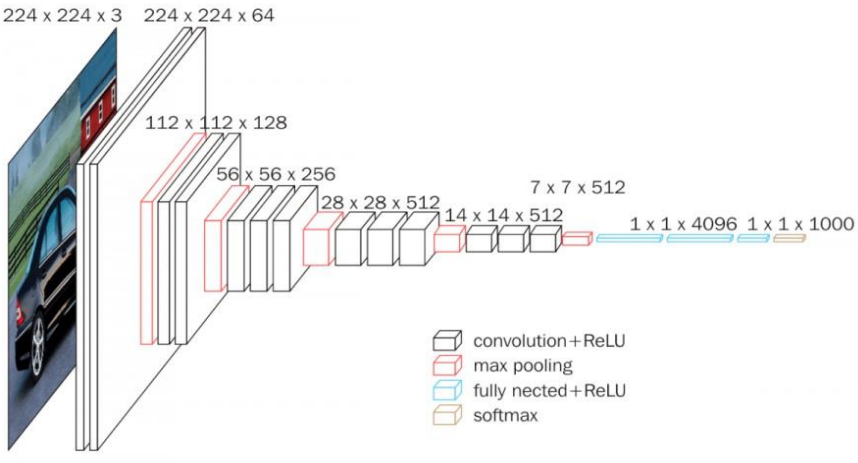


Figure 8.2.1: Network Structure of VGG16

The black box indicates a convolutional layer with ReLU as the activation function. The red box indicates a max pooling layer. Maximum pooling is a feature map pooling procedure that calculates the maximum value in each patch. Max pooling is performed as a method of reducing the computational costs and overfitting by making a more abstracted depiction of the parameters. The blue box indicates a fully connected layer with an activation function of ReLU, and the last fully connected layer uses softmax as the activation function for classification. Input of VGG16 is set to an RGB image of 224x224 size, due to resource limitation the input size is matched to that of 108x108.

The model summary is as follows:

```
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (350, 10, 27, 27, 128)    0
_________________________________________________________________
conv3d_4 (Conv3D)            (350, 10, 27, 27, 256)    884992
_________________________________________________________________
conv3d_5 (Conv3D)            (350, 10, 27, 27, 256)    1769728
_________________________________________________________________
conv3d_6 (Conv3D)            (350, 10, 27, 27, 256)    1769728
_________________________________________________________________
max_pooling3d_2 (MaxPooling3 (350, 5, 14, 14, 256)     0
_________________________________________________________________
conv3d_7 (Conv3D)            (350, 5, 14, 14, 512)     3539456
_________________________________________________________________
conv3d_8 (Conv3D)            (350, 5, 14, 14, 512)     7078400
_________________________________________________________________
conv3d_9 (Conv3D)            (350, 5, 14, 14, 512)     7078400
_________________________________________________________________
max_pooling3d_3 (MaxPooling3 (350, 3, 7, 7, 512)       0
_________________________________________________________________
flatten_1 (Flatten)          (350, 75264)              0
_________________________________________________________________
dense_2 (Dense)              (350, 256)                19267840
_________________________________________________________________
dense_3 (Dense)              (350, 256)                65792
_________________________________________________________________
dense_4 (Dense)              (350, 2)                  514
=================================================================
Total params: 42,234,562
Trainable params: 42,234,562
Non-trainable params: 0
```

Figure 8.2.2: Model Summary of VGG16

8.3 Pre-Trained VGG16 with RNN

VGG16 without any pre-training is computationally more expensive due to its time consumption and needs a large storage requirement. Pre-Training VGG16 overcomes this disadvantage. The layers in pre-trained VGG16 on ImageNet database can be used for feature extraction and detecting generic features. According to the dataset's statistics on ImageNet, the dataset contains some more than 14 million images, over 21 thousand classes, and 1 million images. Here the CNN is the pre-trained VGG16 on the ImageNet database, whereas the RNN is an LSTM. The CNN along with RNN will be able to detect Spatio-temporal features.

Initially, the weights from pre-trained VGG16 on ImageNet Dataset are loaded, the training data is then passed through this base- model to extract generic features. Then this modified dataset is passed through LSTM whose model summary is as follows:

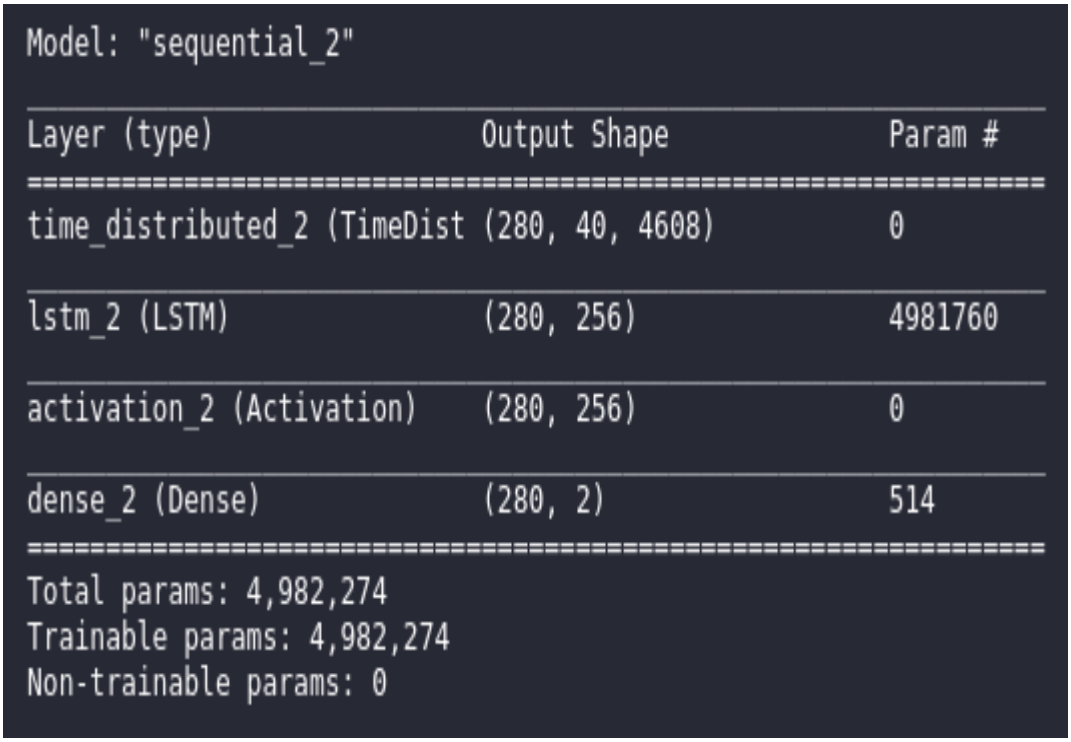


Figure 8.3.1: Model Summary of Pre-Trained VGG16 with Dense Layers and RNN

8.4 Pre-Trained VGG16 and RNN along with Attention Layer

Attention is a technique that is implemented in a neural network to recognize the important parts of the input. Attention gives more importance to certain features. It simply imitates the attention mechanism in the human brain i.e the network uses more computational power to recognize the specific feature from the data. This helps us pay more attention to the fight scenes and not much importance to the background.

Hence the pre-processed video is passed through the pre-trained VGG16 and then passed through the RNN for more temporal features. Since the input is generalized efficiently, it is now easy for the attention layer to search for the specific features and requires less computational power and time.

The model summary is as follows:

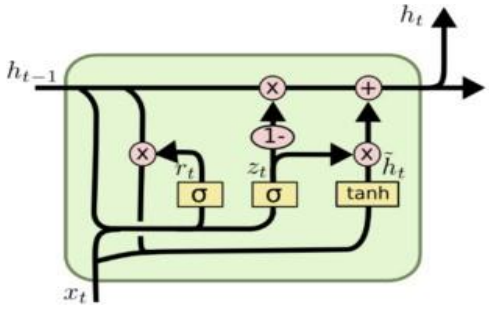
Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 40, 3, 3, 51	0	
reshape (Reshape)	(None, 40, 4608)	0	input_2[0][0]
lstm_4 (LSTM)	(None, 40, 256)	4981760	reshape[0][0]
dense_4 (Dense)	(None, 40, 1)	257	lstm_4[0][0]
flatten_4 (Flatten)	(None, 40)	0	dense_4[0][0]
activation_4 (Activation)	(None, 40)	0	flatten_4[0][0]
repeat_vector (RepeatVector)	(None, 256, 40)	0	activation_4[0][0]
permute (Permute)	(None, 40, 256)	0	repeat_vector[0][0]
multiply (Multiply)	(None, 40, 256)	0	lstm_4[0][0] permute[0][0]
lambda (Lambda)	(None, 256)	0	multiply[0][0]
flatten_5 (Flatten)	(None, 256)	0	lambda[0][0]
dropout (Dropout)	(None, 256)	0	flatten_5[0][0]
dense_5 (Dense)	(None, 2)	514	dropout[0][0]

Total params: 4,982,531
Trainable params: 4,982,531
Non-trainable params: 0

Figure 8.4.1: Model Summary of Pre-Trained VGG16 and RNN along with attention layer

8.5 GRU

Implementing the gating techniques in RNN led to the innovation of Gated Recurrent Units, known as GRU. When compared to LSTMs, GRU is basically a LSTM with yet another forget gate and has fewer parameters. It's used to solve the vanishing gradient problem. This is very helpful because the model can retain all its information from the previous state, which eliminates the risk of vanishing the gradient. Reset Gate, as the name explains itself, resets the past information i.e the amount of information in the previous states that needs to be wiped out and brought back to the initial state. GRU is known to show better performance on smaller datasets.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 8.5.1: GRU Unit

The model summary is as follows:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_1 (TimeDist (280, 40, 34992)          0
_________________________________________________________________
gru_1 (GRU)                  (280, 2)                  209976
_________________________________________________________________
activation_1 (Activation)    (280, 2)                  0
_________________________________________________________________
dense_1 (Dense)              (280, 2)                  6
=================================================================
Total params: 209,982
Trainable params: 209,982
Non-trainable params: 0
```

Figure 8.5.2: Model summary of GRU

8.6 Pre-Trained VGG16 with GRU and Dense Layers

GRU itself cannot suffice the accuracy of the model when we pass a huge input such as the dataset prescribed above. Implementing well-generalized data as an input to GRU and then passing it to the Dense Layers to inhibit the problem of accuracy due to heavy data sets.

GRU itself is an RNN there is no need to pass the data furthermore to another RNN.

The model summary is as follows

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_12 (TimeDis (280, 40, 34992)          0
_________________________________________________________________
activation_12 (Activation)   (280, 40, 34992)          0
_________________________________________________________________
dense_12 (Dense)             (280, 40, 2)              69986
_________________________________________________________________
gru_11 (GRU)                 (280, 2)                  36
=================================================================
Total params: 70,022
Trainable params: 70,022
Non-trainable params: 0
```

Figure 8.6.1: Model summary of Pre-Trained VGG16 with GRU and Dense Layers

8.7 Pre-Trained EfficientNetB6 with RNN

The Google Research Brain team proposed the EfficientNet model in their paper "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks"[5]. In the year 2019, this paper was presented at the International Conference on Machine Learning. In most cases, the models are either too large, too deep or have an extremely high resolution. Increasing these qualities initially helps the model, but it quickly saturates, resulting in a model with more parameters that is inefficient. They are scaled in a more principled manner in EfficientNet. These researchers looked into model scaling and found that adjusting the depth, width, and resolution of the network can assist it to function better. Depending on this analysis,The difference between all 8 efficient nets is that they gradually increased the number of sub-blocks for each efficient net.

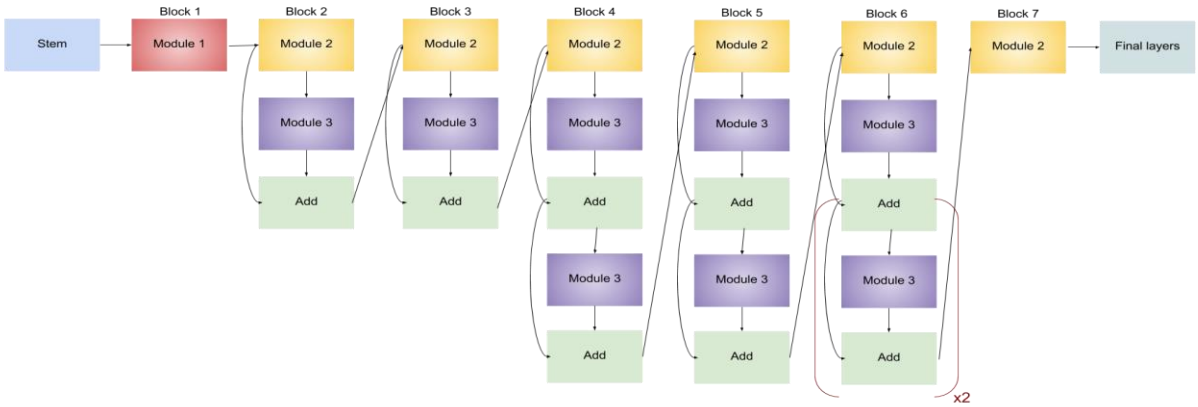


Figure 8.7.1: Structure of EfficientNetB6

The model summary is as follows:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed (TimeDistri (280, 40, 34992)          0
_________________________________________________________________
lstm (LSTM)                  (280, 256)                36094976
_________________________________________________________________
activation (Activation)      (280, 256)                0
_________________________________________________________________
dense (Dense)                (280, 2)                  514
=================================================================
Total params: 36,095,490
Trainable params: 36,095,490
Non-trainable params: 0
```

Figure 8.7.2: Model summary of Pre-Trained EfficientNet with Dense Layers and RNN

CHAPTER 9

CONCLUSION

Our results have shown that a two-phase network works better than a single convLSTM layer with lesser computational overhead. We conclude that two-phase networks offer more flexibility and allow computational overhead to be brought to a minimum, and although classification accuracy is highly dependent on the quality of both spatial and temporal features, the former is far less important.

Future Scope

The use of Transformers for automatically anticipating violence is the future scope of this research on violence detection. Transformers is a deep learning model that directly adopts the properties of the attention layer. Transformers necessarily do not process data in order because the attention mechanism provides the positional argument for any data in the input sequence. This feature provides more parallelization than any other network presented thus far, greatly reducing training time.

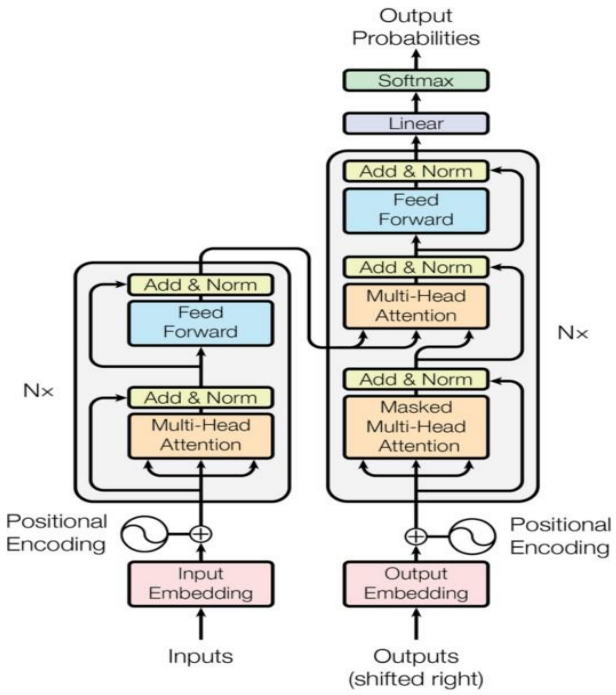


Figure 1: The Transformer - model architecture.

Figure 10.1.1: Transformer Unit

CHAPTER 10

REFERENCES

- Sudhakaran, Swathikiran and Lanz, Oswald. (2017). "Learning to detect violent videos using convolutional long short-term memory." Conference: 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), doi:10.1109/AVSS.2017.8078468.
- Bianculli, Miriana & Falcionelli, Nicola & Sernani, Paolo & Tomassini, Selene & Contardo, Paolo & Lombardi, Mara & Dragoni, Aldo Franco. (2020). A dataset for automatic violence detection in videos. Data in Brief. 33. 106587. 10.1016/j.dib.2020.106587.
- M. Soliman, M. Kamal, M. Nashed, Y. Mostafa, B. Chawky, D. Khattab, " Violence Recognition from Videos using Deep Learning Techniques", Proc. 9th International Conference on Intelligent Computing and Information Systems (ICICIS'19), Cairo, pp. 79-84, 2019
- Simonyan, Karen & Zisserman, Andrew. (2014)." Very Deep Convolutional Networks for Large-Scale Image Recognition". arXiv 1409.1556.
- Tan, Mingxing & Le, Quoc. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.
- [6]D. Moreira et al., "Temporal Robust Features for Violence Detection," 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), Santa Rosa, CA, 2017, pp. 391-399, doi: 10.1109/WACV.2017.50

APPENDICES

SOURCE CODE