

Violence Detection in Videos Using Neural Networks.



By

Raja Nouman Haider

(365163) Fall-2021-MS-Statistics SNS School

Supervisor

Dr. Tahir Mehmood

Department of Sciences

A thesis submitted in partial fulfillment of the requirements for the degree of Statistics

In

,

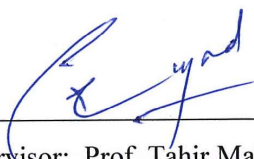
National University of Sciences and Technology (NUST),


Islamabad, Pakistan.


(February 2022)

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS thesis written by **Raja Nouman Haidar** (Registration No. **00000365163**), of **School of Natural Sciences** has been vetted by undersigned, found complete in all respects as per NUST statutes/regulations, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS/M.Phil degree. It is further certified that necessary amendments as pointed out by GEC members and external examiner of the scholar have also been incorporated in the said thesis.

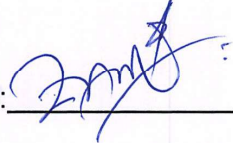
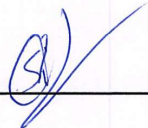
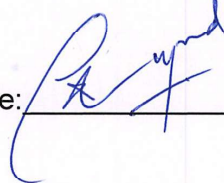
Signature: 
Name of Supervisor: Prof. Tahir Mahmood
Date: 22/12/2023

Signature (HoD): 
Date: 26-12-2023

Signature (Dean/Principal): 
Date: 26.12.2023

National University of Sciences & Technology**MS THESIS WORK**

We hereby recommend that the dissertation prepared under our supervision by: "**Raja Nouman Haidar**" Regn No. **00000365163** Titled: "**Violence Detection in Videos Using Neural Networks**" accepted in partial fulfillment of the requirements for the award of **MS** degree.

Examination Committee Members1. Name: DR. ZAMIR HUSSAINSignature: 2. Name: DR. SHAKEEL AHMEDSignature: Supervisor's Name: PROF. TAHIR MAHMOODSignature: 
Head of Department26/12/2023
Date**COUNTERSIGNED**Date: 26.12.2023
Dean/Principal

Dedication

This thesis is dedicated to all the deserving children who do not have access to quality education especially young girls.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at Department of Sciences at or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Raja Nouman Haider**

Signature: _____

Acknowledgments

Glory be to Allah (S.W.A), the Creator, the Sustainer of the Universe. Who only has the power to honour whom He please, and to abase whom He please. Verily no one can do anything without His will. From the day, I came to NUST till the day of my departure, He was the only one Who blessed me and opened ways for me, and showed me the path of success. There is nothing which can payback for His bounties throughout my research period to complete it successfully.

Raja Nouman Haider

Contents

1	Introduction	1
1.1	Background	2
1.2	Machine learning	3
1.3	Application	4
1.4	Video Surveillance	5
1.5	Machine Learning for Video Surveillance	6
1.6	Objective of Study	7
2	Literature Review	8
3	Methodology	16
3.1	Data Acquisition	16
3.2	Data Processing	17
3.3	Neural Networks and Related Methods	18
3.4	Convolutional Neural Networks	20
3.5	Recurrent Neural Networks	22
3.6	Feature Extraction	23
3.6.1	Visual Geometry Group-16	25
3.6.2	Visual Geometry Group-19	28
3.6.3	EfficientNet-B0	33
3.6.4	Residual Network-50	36

CONTENTS

3.6.5	InceptionV3	41
3.7	Classification Method	44
3.7.1	Long and Short Term Memory (LSTM)	45
3.7.2	Gated Recurrent Unit (GRU)	49
3.8	Activation Functions	51
3.8.1	Rectified Linear Unit (ReLU)	52
3.8.2	Sigmoid Activation Function:	52
3.8.3	Softmax Activation Function:	53
3.9	Model Validation	54
3.10	Performance Evaluation	55
4	Results	58
A	First Appendix	

List of Figures

1.1	Distinction between Supervised And Unsupervised Learning	4
3.1	Hockey fight video dataset consists of 1000 videos out of which 500 videos have violence and other 500 videos have no-violence.	16
3.2	This figure shows the generalized work flow of used methods.	17
3.3	Description of Artificial Neural Network	19
3.4	Description of Convolutional Neural Network	20
3.5	Workflow description of Recurrent Neural Network	22
3.6	Max-pooling operation mathematical description.	24
3.7	Architecture of Visual Geometry Group-16	25
3.8	Architecture of Visual Geometry Group-19	29
3.9	The figure shows the architecture of EfficientNetB0 model.	33
3.10	The figure shows the architecture of ResNet-50 model.	37
3.11	Architecture of InceptionV3	41
3.12	Systematic diagram of Long and Short Term Memory (LSTM)	45
3.13	Systematic representation of Gated Recurrent Unit.	50
4.1	InceptionV3-LSTM method graphical representation of loss and accuracy.	60
4.2	InceptionV3-GRU method graphical representation of loss and accuracy.	60
4.3	EfficientNetB0-GRU method graphical representation of loss and accuracy.	61
4.4	EfficientNetB0-LSTM method graphical representation of loss and accuracy.	61

LIST OF FIGURES

4.5	Resnet50-GRU method graphical representation of loss and accuracy. . .	62
4.6	Resnet50-LSTM method graphical representation of loss and accuracy. .	62
4.7	VGG16-GRU method graphical representation of loss and accuracy. . .	63
4.8	VGG16-LSTM method graphical representation of loss and accuracy. . .	63
4.9	VGG19-LSTM method graphical representation of loss and accuracy. . .	64
4.10	VGG19-GRU method graphical representation of loss and accuracy. . .	64
4.11	This box plot shows the values of accuracy of all the methods at its y-axis and the x-axis shows the method names.	65
4.12	This box plot shows the values of loss of all the methods at its y-axis and the x-axis shows the method names.	66
4.13	This box plot shows the values of sensitivity of class 0 of all the methods at its y-axis and the x-axis shows the method names.	67
4.14	This box plot shows the values of sensitivity of class 1 of all the methods at its y-axis and the x-axis shows the method names.	68
4.15	This box plot shows the values of specificity of class 0 of all the methods at its y-axis and the x-axis shows the method names.	69
4.16	This box plot shows the values of specificity of class 1 of all the methods at its y-axis and the x-axis shows the method names.	70
4.17	This box plot shows the values of precision of class 0 of all the methods at its y-axis and the x-axis shows the method names.	71
4.18	This box plot shows the values of precision of class 1 of all the methods at its y-axis and the x-axis shows the method names.	72
4.19	This box plot shows the values of recall of class 0 of all the methods at its y-axis and the x-axis shows the method names.	73
4.20	This box plot shows the values of recall of class 1 of all the methods at its y-axis and the x-axis shows the method names.	74
4.21	This box plot shows the values of f1-score of class 0 of all the methods at its y-axis and the x-axis shows the method names.	75

LIST OF FIGURES

4.22 This box plot shows the values of f1-score of class 1 of all the methods at
its y-axis and the x-axis shows the method names. 76

List of Tables

3.1	VGG16 Model Architecture	28
3.2	VGG19 Model Architecture	32
3.3	EfficientNetB0 Model Architecture	36
3.4	ResNet-50 Model Architecture (simplified form)	40
3.5	InceptionV3 Model Architecture	44
3.6	LSTM Model Architecture	49
3.7	GRU Model Architecture	51
4.1	Performance of Neural Networks on Hockey fight videos dataset	77
4.2	Performance of Neural Networks on Hockey fight videos dataset	77

Abstract

In this project, we investigate and investigate methods in order to detect violence to totally dismantle the current situation and foresee upcoming trends in investigation of violence. We would offer a complete evaluation of video violence detection challenges outlined in cutting-edge research in our systematic research. This work will explore and identify any outstanding concerns in the current situation or subject, as well as technologically advanced methodologies in video violence detection, data sets for constructing and training video in real time violence frameworks for detection, and debates and identification of outstanding issues. We analyzed 80 research papers chosen among 154 after the Phases of identification, screening, and eligibility in this study. We starts with rapidly explaining the fundamental concept and challenges of video-based violence detection; next, we divide current solutions based on their methodologies, they are divided into three categories. There are traditional methodologies, end-to-end deep learning-based methods, and machine learning-based ways. Finally, we offer and assess video-based violence identification algorithms for testing their efficacy. Furthermore, we describe the open difficulties in video violence identification and assess its future trends. Hockey Fight videos data set has binary classes such as violence and no violence with 1000 videos data, where 800 video's for training and 200 video's for testing the model performance. The process of extracting features, classification, and pre-processing process are the processes that were used in this study. Classification with a success of 93.7 percent accuracy was produced by VGG16-LSTM, which outperformed 93.4 percent from VGG16-LSTM, 92.3 percent from EfficientNetB0-GRU, 92.1 percent from VGG19-GRU, 90.9 percent from VGG19-LSTM, 90.4 percent from InceptionV3-LSTM, 90 percent from EfficientNetB0-LSTM, 89.1 percent from InceptionV3-GRU, 83.9 percent from Resnet50-LSTM and 82.6 percent from Resnet50-GRU for the classification of Hockey fight videos. According to other evaluation performances such as sensitivity,

LIST OF TABLES

specificity, f1-score, precision, recall, and loss. VGG16-GRU outperforms all the other methods used in this research. The findings of this research indicate Neural Network models specifically VGG16-GRU show significant potential for accurately and efficiently detecting violence.

Keywords: Violence detection, Machine learning, Deep learning, Computer vision, Artificial intelligence, Video features, Data sets

CHAPTER 1

Introduction

Public violence has substantially escalated in recent years. Both in the streets and high schools. As a result, surveillance cameras are now widely used. This has made it easier for the government to recognize these incidents and take appropriate action. However, practically all systems now rely on manual video inspection to find such events, which is essentially wasteful. Therefore, a system that can automatically monitor and recognize the surveillance videos is essential. Large data sets and processing power have enabled the development of a range of deep learning algorithms, resulting in a paradigm change. Computer vision is a field that is growing in popularity. To do so we handle issues including object detection, recognition, tracking, and action, various strategies have been created.

Surveillance and anomaly detection have grown increasingly as crucial as the video data volume is increased significantly. Such anomalous occurrences are unusual in comparison to normal activities. As a result, in order to avoid work and duration loss, building automated video surveillance systems have emerged to detect anomalies as necessary. It is difficult to detect abnormalities in films because term "anomaly" is frequently vague and defined poorly. They vary substantially based on settings and situations where they take place. Riding a bike on a regular path is one example of ordinary, nevertheless it should be emphasized that doing so in a walk-only lane is unusual. Anomalous behavior is distinguished by unequal internal occlusion, which is difficult to explain. Now general overview of what the proposed thesis entails. Remember that it is not simply a description of the contents of each part. Moreover, video data encoding and modeling are more complex due to its high dimensionality, resolution, noise, and rapidly changing events

and interactions. According to Yazdi and Bouwmans (2018), further changes in lighting, perspective adjustments, camera motions, and so on are among the difficulties. Violence identification is a critical component of video based violence detection. Because of rise in security concerns around world, monitoring via video cameras individually has become critical, and early discovery of these violent behaviors may considerably reduce the hazards. The primary goal of a system for detecting violence is to detect anomalous conduct that falls under the category of violence. When the behavior of an event differs from what is expected, it is labeled violent. Such abnormalities include someone punching, kicking, pulling another person, and so on. An item at an odd location, strange patterns of movement such as disordered movement, sudden movements, and All of these signs point to violent situations. Because human supervision for full video stream is not feasible due to labor's repetitious nature and time required, real-time automatic detection of violent occurrences is critical to preventing such situations. Many researchers investigated various strategies for improving performance in detecting violence. This thesis examines and addresses numerous strategies for identifying violence from surveillance camera recordings using a complete review of the literature. The main objective of this research is to give a comprehensive examination of strategies for identifying violence. Various strategies for identifying aggressiveness and violence conduct in video have been created over last decade. These methods must be categorize, examined, and summarized.

1.1 Background

In the seventeenth century, when machine learning first arose, Pascal and Leibniz created computers that could simulate human addition and subtraction. In the contemporary period, IBM's Arthur Samuel created the phrase "machine learning" and proved that checkers could be taught to computers. Rosenblatt then created the perceptron, one of the first neural network topologies, in 1958. Werbos' invention of the multilayer perceptron (MLP) in 1975 was a significant step forward. In 1986, Cortes and Vapnik developed support vector machines, and Quinlan developed decision trees. Deep learning algorithms with distributed multilayered learning have recently been created.

In the 1950s, the assumption that machines could display human comprehension gave rise to the term "artificial intelligence." According to Jerrold S. Maxmen, artificial intelligence

(AI) will usher in the twenty-first century. ML, a subcategory of artificial intelligence (AI), exemplifies the empirical acquisition associated with human cognition and may be studied and improved using computer algorithms. The computer may take input and estimate a result through repeats and algorithm tweaks. As a result, in order to increase its ability to foresee future events, ground truth is constantly adjusted while attempts are made to compare the results with the algorithm's outputs. Machine learning helps with data interpretation in numerous fields, including research, medicine, the economics, and policy making. Machine learning is implemented using equations from mathematics, statistical analysis, and computer programming. Machine learning is classified as follows: 1. supervised machine learning, and 2. unsupervised machine learning.

Several strategies for analyzing data are referred to as "statistical learning." These tools are classified as either supervised or unsupervised. The overarching supervised goal of statistical learning is to construct a statistical model capable of calculating or anticipating an output based on one or more inputs. This type of issue affects a wide number of professions, including business, medicine, astrophysics, and policymaking. Although unsupervised statistical learning has no supervised outputs, we can nonetheless draw correlations and patterns from such data.

1.2 Machine learning

Within artificial intelligence (AI) and computer science, machine learning leverages data and algorithms to simulate human learning processes, progressively increasing its precision. Machine learning is a large field that includes a variety of academic specialties, such as information and technology, statistics, probability, artificial intelligence, psychology, neurology, and many more. Machine learning can be used to swiftly solve problems by building a model that faithfully captures a selected dataset. By instructing machines to imitate the functioning of the machine learning and the human brain have increased the scope of statistics research and made it is all-inclusive field that produces fundamental computational statistical models of learning processes. The creation of algorithms that allow computers to learn is at the heart of machine learning. Finding patterns in data, whether statistical or otherwise, is a learning process. The main purpose of developing goal of machine learning algorithms was to be able to mimic how people learn specific skills. The difficulty of learning in varied contexts can also be revealed by

these algorithms. Machine learning and Big Data computing technologies are evolving in a different way than in the past. Lately, machine learning has developed the capacity for automation apply a wide range of complex mathematical computations to massive amounts of information, enabling much high calculation speed of outcomes. Several machine algorithms learning are created, maintained, and enhanced up till today. The use of adaptation in programming is fairly widespread. It is utilized in machine learning systems that are able to recognize models, learn from past errors, extract current information from data, and increase the output's productivity and accuracy. More than ever, machine learning uses multidimensional data that are present in several application domains.

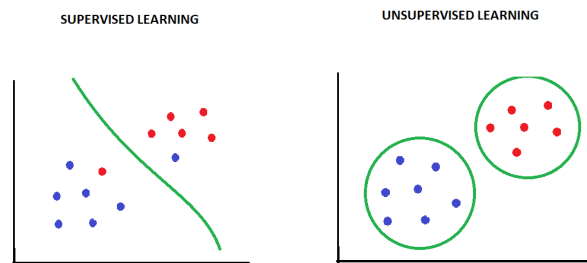


Figure 1.1: Distinction between Supervised And Unsupervised Learning

Two types of models for supervised learning are classification and regression. When the response variable is made up of continually actual values, such as time, money, intensity, length, and so on, regression models are used. It is useful to estimate the link between the numerical value data of an outcome variable and many explanatory variables. The categorization model, on the other hand, is a type of directed learning in which the response variable is classified as 'child' or 'adult,' 'True' or 'False,' 'male' or 'female,' or binary values '0' or '1'. Real-world examples include test scorecard prediction, sentiment analysis, and light detection.

1.3 Application

Machine learning is extremely important in the field of AI research. Although an intelligent system cannot be described without the ability to learn, the former intelligent system often lacks this feature. It has been applied in several of artificial intelligence (AI) fields, including computer vision, natural language understanding, additional reasoning,

intelligent robotics, and model recognition. Search engines, biological diagnosis, credit card fraud detection, stock market investigation, DNA sequencing sequences, handwriting and voice matching identification, gaming approach, Action recognition and robotics utilization are a few instances of specific applications.

1.4 Video Surveillance

Regarding the domain of computer vision, machine learning and action recognition that focuses on discovering and comprehending human acts or activities from digital videos. It involves developing algorithms and models to automatically recognize and classify different actions or movements performed by humans in video sequences.

Action recognition has numerous practical sports analysis, human-computer interaction, video surveillance, healthcare monitoring, and entertainment (e.g., gesture-based gaming) are some of the applications.

Action recognition algorithms require large datasets of labeled video clips where each clip contains one or more actions. These databases are used for training and test system of action recognition models.

Features: Feature extraction is a crucial step in action recognition. Various attributes, including 3D trajectories, optical flux, or deep-learning-based representations, are used to record movement and appearance data in video frames.

With the introduction of deep neural networks, especially Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), deep learning-based models have become dominant in action recognition. These models can learn complex spatiotemporal representations directly from video data. Temporal Modeling: Recognizing actions often requires understanding the temporal order of a video sequence's frames. Recurrent neural networks (RNNs), Long Short-Term Memory (LSTM) networks, and 3D Convolutional Neural Networks (3D CNNs) are commonly utilized to create temporal dependencies.

Action recognition faces several challenges, including variations in lighting, viewpoint, scale, occlusion, plus the existence of multiple people or objects in the image. Handling certain difficulties requires robust feature extraction and modeling techniques. Some applications, such as gesture recognition for human-computer interaction or action

recognition for robotics, require real-time action recognition systems that can process video streams in near real-time. Action recognition of a number of metrics, including recall, accuracy, precision, F1-score, and mean Average Precision on test datasets. Cross-validation and action-specific metrics are also used in some cases.

1.5 Machine Learning for Video Surveillance

Machine learning has significantly transformed video surveillance systems by enabling more advanced and intelligent functionalities. Here are some key ways in which machine learning is applied in video surveillance.

Machine learning models, notably convolutional neural networks (CNNs), are used to identify and track objects of interest within video streams. This can include identifying and following people, vehicles, or specific objects within a scene.

Facial recognition algorithms are employed to identify and track individuals within video footage. This is useful for security applications, access control, and tracking the movement of specific individuals.

Machine learning models are trained to recognize normal behavior patterns in video data. Any deviation from these patterns is flagged as an anomaly, potentially indicating suspicious or unusual activity. Anomaly detection identification can be critical in security and monitoring potential threats.

Machine learning is used to classify objects or actions within video feeds. For instance, it can distinguish between different types of vehicles (cars, trucks, bicycles), recognize gestures or hand signals, or identify weapons.

Machine learning models can analyze human behavior within video footage. This includes tracking movement patterns, identifying interactions between people, and detecting aggressive or abnormal behavior.

License Plate Recognition systems use machine learning to recognize and read license plates on vehicles. This is widely used in traffic monitoring, toll collection, and parking management.

Machine learning can help analyze crowd dynamics, count how many persons are in a scene, and detect crowd density. This is important for event management, public safety,

and urban planning.

Gesture recognition algorithms can identify and interpret hand gestures or body movements in video streams, making it useful for human-computer interaction and user interface control.

Machine learning can be used to recognize specific events or actions within video footage, such as vehicle collisions, people falling, or package thefts.

By analyzing historical video surveillance data, machine learning can help predict potential security breaches or incidents based on patterns and trends.

Machine learning can be used to automatically blur or obscure faces and sensitive information in video feeds to protect individuals' privacy.

When unusual or suspicious activities are detected, machine learning systems can trigger automated alerts or notifications to security personnel or relevant authorities.

To develop a full security ecosystem, machine learning can be connected with other security systems such as access control, alarms, and intercoms.

Machine learning allows for the Video surveillance systems' scaling. It can handle massive amounts of video data more efficiently than manual monitoring.

1.6 Objective of Study

- . To detect the violence in videos so that we can find the best ways to overcome the violence situation in our daily life.
- . To assess the performance of several neural networks in identifying violence in hockey fight video dataset.

CHAPTER 2

Literature Review

Image processing techniques is an effective way for Violence detection in hockey fight videos and it has become quite influential in the recent years of technological advancement. Using Lagrangian theory in computer vision, offers an extensive collection of techniques for assessing long-term data. The author suggest a tailored Lagrangian technique based on this principle, for the automatic recognition of violent scenes in video footage. [26]. These researchers present a new feature that utilizes a spatio-temporal framework that considers Lagrangian direction fields along with appearance, long-term motion data, and background movement corrections. They make use of an extended bag-of-words technique in late-fusion as a per-video categorization approach to ensure proper temporal and spatial feature sizes. The "Hockey Fight" project included three datasets in which experiments were conducted. [6] Less attention has been dedicated to the determination of violent acts or arguments in the majority of studies on action recognition, which has focused on identifying individuals and surveillance, idling, among other comparable behaviors. Although local spatiotemporal feature extractors have been studied previously, they suffer from the need for complex optical flow computations. Although the quicker one is the temporal derivative method than optical flow. It yields a result that is scale-dependent and of low precision when used in isolation. [28] Action recognition is becoming a relevant study area in the realm of computer vision. Nonetheless, most studies have focused on relatively simple actions such as clapping, walking, and running. Much less effort has been paid to the identification of particular events that have immediate practical implications, such fighting or other aggressive behavior. In some places, like prisons, mental health centers, or even video cameras on phones monitoring can be quite helpful. Gracia suggested a fresh method

to spot violent sequences. Features extracted from motion blobs are utilized to distinguish between fighting and non-fighting events. Proposed technique was tested on three distinct datasets, including the "Movies" dataset, which had 200 video clips. [6]. Local Motion technique, along with its v-1 and v-2 variations that make use of Random Forest, AdaBoost, and KNN classifiers. Even if the suggested method is not as effective as it could be, it is appropriate for real-world applications because of its significantly faster calculation time. In video surveillance environments like stations, schools, and health buildings, it's imperative to instantly recognize hostile activities. However, prior identification algorithms often extracted descriptors around spatiotemporal interesting regions or statistical features in motion zones, which led to limited capabilities in effectively identifying violent events in videos. [36]. while most research has concentrated on activity recognition, combat detection has gotten far less attention. Ability might be quite useful. Most strategies require domain expertise to construct complex handmade qualities from inputs. Conversely, deep learning techniques can work directly with unprocessed inputs and automatically extract pertinent features. Ding thus developed a novel 3D ConvNets technique for detecting video violence that doesn't require any prior knowledge. [13]. To evaluate the approach, experimental validation was performed on the "Hockey fights" dataset [6]. The findings indicate that the strategy performs superior to manual features. Campus violence is the most serious form of bullying that occurs in schools and is a global social concern. More options, including video-based solutions, are available for identifying college violence using AI and remote surveillance are being used to combat technologies advancements. [40] use auditory and visual data to detect campus violence. Role-playing is employed to gather data on campus violence, and feature vectors with 4096 dimensions are retrieved from each of the 16 video frame. 3D CNN has an overall precision of 92.00 percent and is used to extract features and classification. MFCCs as acoustic characteristics are extracted from three speech emotion datasets: CASIA dataset [3], which contains 960 samples, Finnish emotional dataset and Chinese emotional dataset. To address the issue of evidence dispute, an improved D-S algorithm is developed. Consequently, the accuracy of recognition rose to 97%

The goal of the NetVLAD architecture is to solve the problem of widespread visual place recognition by rapidly and precisely determining the position of a given query image. A CNN-based method for unsupervised location recognition is called NetVLAD. This work makes three primary contributions. Firstly, a CNN architecture that can

be programmed end-to-end is designed for the location recognition issue. The main element of the technique is NetVLAD, a unique generalized VLAD layer that draws inspiration from the popular picture format "Vector of Locally Aggregated Descriptors (VLAD)." Any CNN model can easily add the layer and train it using backpropagation. The following contribution is the creation of an instructional technique for end-to-end learning of architectural variables using Google Street View Time Machine photographs of identical locations throughout a period of time which is centered on a unique weakly supervised ranking loss. Lastly, the authors showed that on two challenging place identification benchmarks, the suggested architecture performs better than both commercially available CNN descriptors and non-learned picture representations, as well as contemporary state-of-the-art picture representations using common standards for image retrieval, using the Pittsburgh [11] and Tokyo 24/7 [11] datasets. [33] offer a technique for detecting violence in real time that analyzes massive quantities of data that is streaming and understands hostility utilizing a simulation of human intelligence. Spark framework is used to evaluate the massive amounts of real-time video that are received by the system from various sources. The frames are split apart, and the Spark framework's HOG function is used to extract each frame's properties. Following that, the frames are grouped according to attributes that have been developed with the BDLSTM network for the recognition of violent scenes, like the negative model, the human component model, and the violent model. Data access is possible in either direction thanks to the bidirectional LSTM. Consequently, the result is produced considering both historical and prospective data. The (VID), which comprises 2314 videos with 1077 fights and 1237 no-fights, is used to train the network. A dataset of 409 violent video episodes and 410 video episodes with neutral scenes was also produced by the authors. The model's 94.5 percent detection accuracy validates its effectiveness and highlights the robustness of the system. [22] offer a method for identifying violent scenes according to auditory information from videos. CNN can be used in the role of a classifier or an extractor of deep acoustic features. To start, the 40-dimensional (MFB) is the input feature used by the CNN. After then, the footage is split up into smaller parts. To examine three feature maps comprise local features and MFB features. Next, features are represented using CNN. SVM classifiers are constructed using characteristics derived from CNNs. Next, the violent scene identification method is used for every frame of the video. Next, maximum or minimum pooling is applied at the segment level to generate detection.

Investigations are conducted on the MediaEval dataset [12] and the outcomes show that the proposed method surpasses the core strategies. Handcrafted techniques was published [41] These qualities are connected to appearance, gait, and representational images; a CNN receives them as spatiotemporal, temporal, and spatial inputs. A novel approach to deep violence detection that takes advantage of the unique characteristics of l stream. With every frame, the neural network is trained by the spatial stream to identify trends in its environment. The temporal stream uses a changed differential magnitude of optical movement to become familiar with motion patterns of aggressive conduct over a span of three consecutive frames. In order to better understand aggressive behaviors, the scientists also developed a characteristic of the spatiotemporal stream that discriminates, utilizing a novel differential motion energy picture. This method incorporates several components of aggressive actions by merging the findings of several streams. The projected CNN network underwent training on three datasets: Hockey [18], Movie [6] and ViF [17]. Regarding precision and processing duration, the strategy outperformed existing alternatives. A deep neural network was proposed by [27] to identify violent scenes in videos. Frame-level characteristics are extracted from a video. The LSTM, which makes use of a convolutional gate, is then used to gather the frame-level attributes. Local motion in the video can be analyzed thanks to the CNN's for it's capacity to record specific spatiotemporal features locally in conjunction with the ConvLSTM. The paper also suggested forcing the model to encode the changes in the video by feeding it surrounding frame differences as input. On three widely used benchmark datasets, including "Hockey" [18], "Movies" [6], and "Violent-Flows" [31], the provided feature extraction approach is evaluated for recognition accuracy. The results were contrasted with those obtained utilizing cutting-edge techniques. It was discovered that the suggested strategy outperformed leading state-of-the-art algorithms such as three streams + LSTM, ViF, and ViF+OVIF in the promising domain of violent film identification. The Mask RCNN and LSTM ensemble model was developed to figure out violent actions of an individual [38] Human masks and crucial points were taken out first, followed by temporal data. Experiments were carried out in datasets from Weizmann [2], KTH [1], and our own. The outcomes demonstrate the fact that the suggested model works superior to standalone models, with the top result being 93.4 percent accuracy in identifying violent behavior. The suggested method is more industry-relevant, which enhances security for society. Conventional methods focus on subjective attributes that

might not be adequately discriminative to identify violent behavior. We provide a unique method to identify human violent behavior in movies that combines trajectory and deep CNN and leverages both hand-made and extensively trained features, motivated by the strong performance of deep learning-based approaches. [24]. To evaluate the proposed strategy, tests are run on two distinct datasets for violence: "Hockey Fights" [6] and "Crowd Violence" [34]. The outcomes reveal that the suggested methodology surpasses the most recent approaches like as HOG, HOF, ViF, and others on these datasets. Using a bidirectional ConvLSTM module to capture pertinent spatiotemporal data and a changed 3D DenseNet for an array of heads self-attention layer, [39] outline a novel method for figuring out if a movie contains a violent scenario or not. Additionally, an analysis of ablation is performed on the input frames to compare the effects of the attention layer and neighboring frame removal with dense optical flow. The results show that integrating optical flow with the mechanism of attention can improve outcomes by as much as 4.4 percent. Study employed four datasets and illustrated superior performance as opposed to cutting-edge techniques. Specifically, the number of network parameters required was reduced to 4.5 million, and the test accuracy increased from 95.6 percent on the most complex dataset to 100 percent on the simplest. Additionally, the inference time was less than 0.3 seconds for the longest clips. In computer vision, human action recognition has become a major field of study. Less emphasis has been paid to violent conduct or fights, although they can be beneficial in a range of surveillance footage scenarios, as prisons, mental health facilities, or even on a private mobiles. The development of violence or fight detectors is prompted by their diverse variety of applications. Efficiency is the main characteristic of the detectors, hence these techniques should have a quick computational turnaround. While hand-crafted spatio-temporal characteristics yield very accurate look and motion, the price of extraction individual characteristics are still too high for most sensible uses. Applying deep learning paradigm to a task is done for the first time using a 3D CNN, taking in complete using a video feed as input. Nevertheless, motion features are essential to our work, because full video input introduces noise and duplicates learning. A "handcrafted/learned" framework with hybrid characteristics was created with this goal in mind [29]. The method looks for an example picture from the video series that is utilized as a classifier for Hough forest, as an input for feature extraction. After that, 2D CNN is used to classify the picture in order to ascertain the end of the order. Three sets of data for detecting violence have

been employed for the studies: "Movie" [18], "Hockey" [6], and "Behavior" [32]. The results indicate that the recommended technique surpasses the different deep learning and traditional techniques in terms of standard deviations and accuracy. Both an SVM classifier and a two-stream CNN architecture are suggested [30]. The three components of the approach are label fusion, training, and feature extraction. An already-trained Imagenet VGG-f architecture is used by each stream CNN. Whereas the second stream collects motion data, the first stream uses consecutive frame differences to obtain visual information. Then, sight and motion data are used to train two SVM classifiers. Ultimately, a combination of labels approach is accustomed to produce the outcome of the detection. This strategy's main benefit is its ease of implementation. However, identifying violence in large groups is difficult since this approach is not sensitive to violent acts among people at close range. Accattoli [37] employed two-stream CNNs in an analogous manner. In order to obtain extended temporal data, they propose integrating improved trajectory CNN. They extract geographical and chronological data using two networks of VGG-19. Spatial data is recovered using visual frames, as well as temporal data is recovered using dense optical flow pictures. In educational institutions, medical centers, and other monitoring domains, a stronger Security systems are necessary to identify individuals that are violent or deviant activities in order to avoid any deaths that might cause harm to the environment, the economy, or society. For this objective, [35] provides a three-stage complete profound understanding of violence detection solution. To reduce and get past the needless handling of meaningless sometimes, individuals are initially detected in the security camera feed with a condensed CNN model. Secondly, a 16-frame series of acknowledged persons are fed into 3D CNN, which retrieves spatial properties from the sequences and provides the Softmax classifier with them. The authors additionally enhanced the 3D CNN model by utilizing Intel's neural network optimization and open visual inference tools, which turn transforming the training model into a halfway representation and modify for the best possible performance at the final platform for the ultimate prognosis of aggressive conduct. When hostile conduct is found, a notification is delivered to the nearest police department or security firm in order that necessary safeguards can be taken. In the trials, the datasets "Violent Crowd" [7], "Hockey" [6], and "Violence in Movies" [18] are employed. In terms of accuracy, the experimental results reveal that the suggested method works better than cutting-edge algorithms[25]. Among the deep learning architectures ConvNets is the most extensively

utilized. [9] trained deep ConvNets on sufficiently big picture datasets of over 15 million tagged images first. ConvNets are widely employed in different pattern recognition domains due to their outstanding results [16]. When compared to traditional machine learning approaches and its handcrafted features, ConvNets are able to pick up some symbolic qualities automatically. Mo utilized ConvNets promptly both a multilayer perceptron and feature extraction was employed for categorization. One issue with using deep learning for HAR is finding out How to use it to tiny datasets, which are often less than what ConvNets require. Generating or dropping more training examples is a common strategy, as is shifting HAR to a classification of a still picture issue to take advantage of large picture datasets (e.g., ImageNet) to pretrain the ConvNets. Wang developed three methods for employing ConvNets with little training sets of data. To begin, 3D depth map points are turned to imitate several perspectives, and WHDMMs are built at various temporal scales. Finally, before being input into the ConvNets, various patterns of motion are improved and converted into channels that mimic RGB. In contrast, Simonyan and Zisserm [14] employ a large picture ConvNets were pre-trained on the dataset. They looked into a two-stream architecture, with the spatial stream including appearance data from motionless frames and applied through a spatial stream ConvNet. The spatial ConvNet is pretrained on a large image classification dataset because it is an image classification architecture. Long-range dynamics data is essential and should be clearly represented, according to Li et al. [20]. As a consequence, they presented VLAD3, a representation that captures not just with ConvNets for short-term dynamics, but also medium- and long-term dynamics using dynamic systems that are linear and the VLAD descriptor. Wang [20] introduced TDD that merged hand-crafted local features (e.g., STIP, improved trajectories) with deep-learned features (3D ConvNets [5],[8], two-stream ConvNets [14]). The suggested TDD combines the benefits of these two aspects and employs cutting-edge enhanced paths and dual-stream ConvNets. DNNs, unlike ConvNets, continue to use hand-crafted features rather than deep networks automatically learning features from raw data. Berlin and John employed Interest points based on Harris corners and features based on histograms as input [21]. To recognize human-human interactions, the suggested several auto encoders stacked on a deep neural network is deployed. Huang [23] discovered Lie group features by integrating a deep network design with a Lie group structure. Because it can hold observations in memory for extended periods of time, the LSTM architecture is the most common

among RNN designs [10]. As early research for activity identification, an LSTM network served as a categorize events in soccer footage [4]. Then, another research [10] explicitly confirmed LSTM's stability even in test situations deteriorated, indicating its capacity for strong recognition in the actual world. Veeriah et al. [19] expanded the LSTM to differential RNNs. The most important spatiotemporal models of activities are learnt by calculating various levels of derivative of state that are responsive to the spatiotemporal structure, although typical significant dynamic patterns of activity are not captured by LSTM. RNNs may be utilized for identifying activity in both skeletal data and videos. Du et al [15] established a hierarchical RNNs framework for skeleton-based identification.

Methodology

3.1 Data Acquisition

The Hockey Fight video dataset consisting of 1000 videos out of which 500 videos have violence and other 500 videos have no violence. The data is acquired from kaggle and link below for the data is given below.

<https://www.kaggle.com/datasets/yassershrief/hockey-fight-videos>

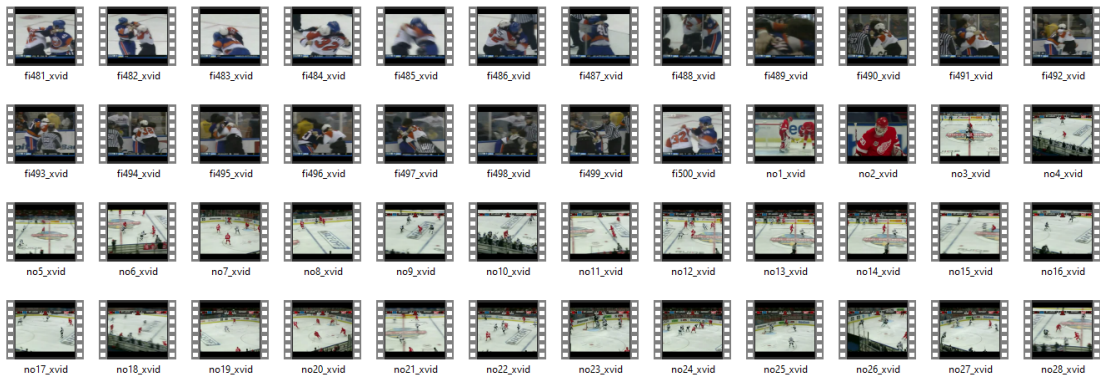


Figure 3.1: Hockey fight video dataset consists of 1000 videos out of which 500 videos have violence and other 500 videos have no-violence.

Public violence has substantially escalated in recent years. Both in the streets and high schools. As a result, surveillance cameras are now widely used. This has made it easier for the government to recognize these incidents and take appropriate action. However, practically all systems now rely on manual video inspection to find such events, which is essentially wasteful. Therefore, a system that can automatically monitor and

recognize the surveillance videos is essential. Large data sets and processing resources have made it possible for the creation of several deep learning techniques, which has led to a revolutionary shift in the area of vision technology. In order to handle issues including object detection, recognition, tracking, and action, various strategies have been created. The methods that we are going to use for the violence detection of HOCKEY FIGHT dataset are VGG-16, VGG-19, Inception-V3, EfficientNet-B0, ResNet-50, LSTM and GRU. we can see the work flow in Figure 3.2.

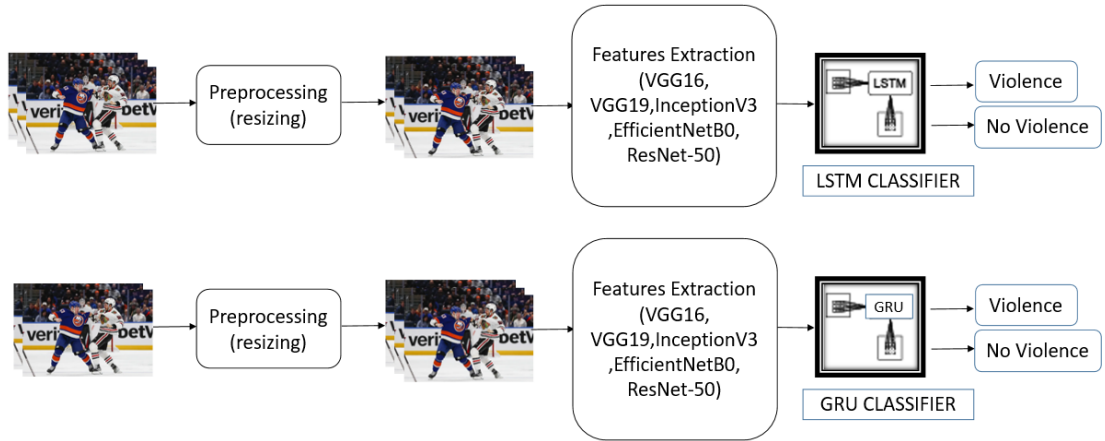


Figure 3.2: This figure shows the generalized work flow of used methods.

The data processing for our dataset is give below.

3.2 Data Processing

Resizing Images using to Bicubic Interpolation. Bicubic interpolation is a method for resizing images that involves weighted averaging of neighboring pixel values. The mathematical equation for bicubic interpolation can be quite complex and depends on the specific interpolation kernel used. Here, I'll provide a simplified representation:

$$P_{\text{resized}}(x, y) : \text{Position of the resized image's pixel value}(x, y).$$

$$P_{\text{original}}(x', y') : \text{Pixel values of neighboring pixels in the original image.}$$

$W(x', y')$: Interpolation weights for neighboring pixels.

In bicubic interpolation, we calculate $P_{\text{resized}}(x, y)$ by summing up the contributions from neighboring pixels ($P_{\text{original}}(x', y')$) weighted by the corresponding interpolation weights ($W(x', y')$) at each position (x, y) in the resized image.

The specific mathematical equations for interpolation weights and their calculation may vary based on the exact implementation and kernel used for bicubic interpolation.

Pixel value normalization is a straightforward mathematical operation:

$P_{\text{normalized}}(x, y)$: Normalized Position of the resized image's pixel value (x, y) .

$P_{\text{original}}(x, y)$: Original Position of the resized image's pixel value (x, y) .

The normalization operation scales each pixel value ($P_{\text{original}}(x, y)$) by a constant factor, which in this case is 255, to ensure that the resulting pixel values are in the range [0, 1]. The mathematical equation for pixel value normalization is:

$$P_{\text{normalized}}(x, y) = \frac{P_{\text{original}}(x, y)}{255}$$

This equation divides each pixel value by 255, thereby scaling them to the desired range.

3.3 Neural Networks and Related Methods

Artificial neural networks (ANNs), sometimes referred to as neural networks or simply "neural nets," are a fundamental component of machine learning and artificial intelligence. They are motivated by the composition and operations of the human brain, where interconnected neurons work together to process and transmit information. Neural networks consist of hierarchically arranged layers of artificial neurons, or nodes to perform various computational tasks. Here are few important neural network aspects:

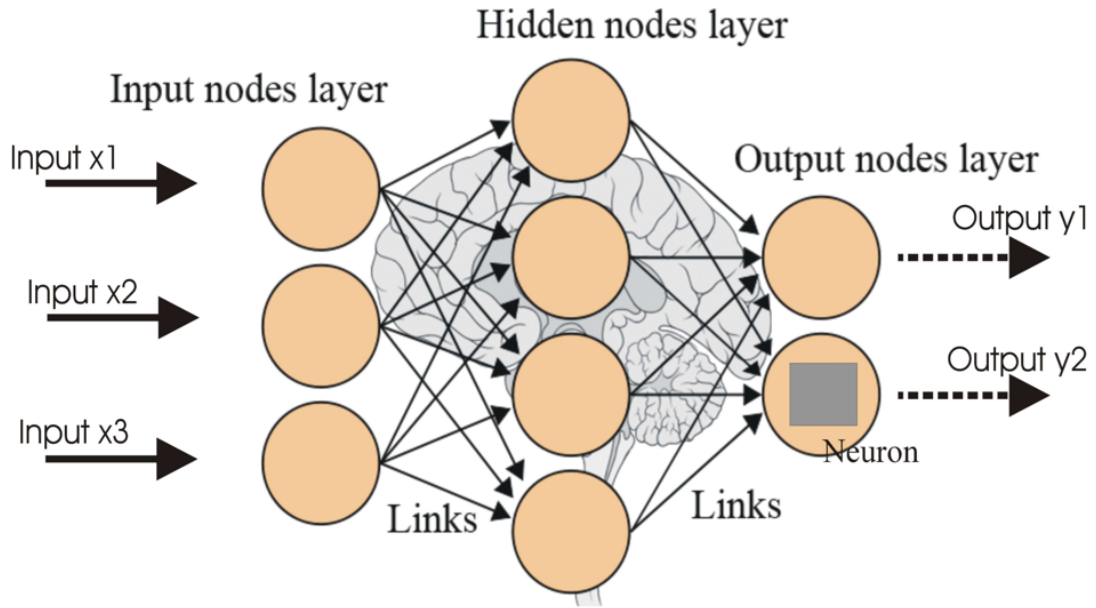


Figure 3.3: Description of Artificial Neural Network

Neurons are the fundamental processing units the neural network's. Every neuron is fed information, processes it using a set of weighted connections, applies an activation function, and produces an output.

Neural networks are typically organized into layers: an input layer, an output layer, and one or more hidden layers. Data is received by the input layer and passed to the hidden layers, which perform complex computations. Using the output layer, the final prediction or result.

The connections that neurons make are represented using weights. These masses establish the power of the connections and are adjusted during the training process. A neuron's weighted sum of inputs is calculated and passed through a function for activation that produces the result of the neuron.

Neural network can represent intricate relationships in data thanks to the non-linearity that activation functions bring to the system. Sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU) are examples of common activation functions, and variants like Leaky ReLU.

3.4 Convolutional Neural Networks

Deep learning neural networks of the Convolutional Neural Networks (CNN) class have shown remarkable efficacy in a variety of computer vision uses, such as picture production, object detection, and categorization. CNN were inspired by the visual processing of animal brains and have transformed the field of computer vision. Here are some key components and concepts associated with CNN as shown in figure 3.3:

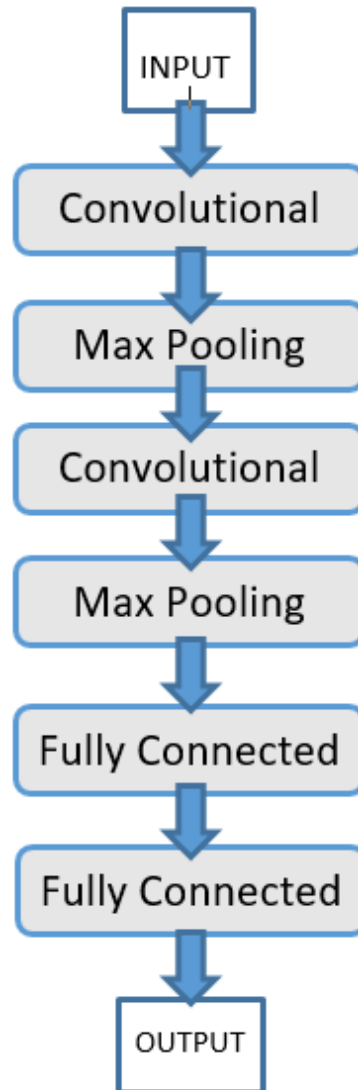


Figure 3.4: Description of Convolutional Neural Network

CNN are built upon Convolutional layers are in charge of learning spatial feature hierarchies from incoming data. These layers make use of convolution operations to input

images, using a number of teachable filters (sometimes also known as kernels or weights). In the convolution operation helps with the data the detection of edges, local patterns, and basic features.

Using pooling layers frequently to lower the spatial dimensions of the feature maps, but keeping the crucial information intact, following convolutional layers. CNNs frequently employ the pooling techniques of max-pooling and average-pooling.

Output of convolutional and pooling layers is subjected to activation functions that are not linear, such as ReLU, which impart non-linearity to the network and facilitate its learning of complicated representations.

At least one completely linked layer is usually employed after the convolutional and pooling layers to perform classifications or predictions based on the learnt features. Neuron in these levels connects to each and every other neuron within the layers above and below.

A one-dimensional vector is often created by flattening the feature maps prior to connecting to fully connected layers to be processed by the dense layers.

A regularization method called dropout is employed to stop overfitting. In order to compel the network to acquire more resilient properties, it randomly removes a portion of neurons during training.

Filters or kernels are small windows that slide over the input data, capturing regional patterns. These filters are acquired through instruction and become specialized to detect certain features, such as edges, textures, or higher-level shapes.

Stride determines how the convolutional filters move through the input data. While a short stride retains more spatial information, a bigger stride decreases feature maps' spatial dimensions.

To ensure that, regulate spatial dimensions of feature maps following convolution, padding is frequently added to the input data. Zero padding adds zeros around the input to maintain the same spatial dimensions.

CNNs consist of multiple convolutional layers stacked above one another. The deeper layers learn increasingly abstract and complex features by combining information from earlier layers, creating a hierarchy of features. Pretrained CNN models, such as VGG, ResNet, Inception, and MobileNet, are taught using extensive image databases such as

ImageNet. These models have learned rich feature representations and can be modified to suit particular needs or applied as feature extractors in transfer learning.

3.5 Recurrent Neural Networks

One type of artificial neural network is recurrent neural network (RNN), which have internal memory states that allow them to handle sequential data. In contrast to feed-forward neural networks, which utilize fixed-size input vectors, RNNs are ideal for jobs requiring data sequences, including speech recognition, natural language processing, time series analysis, and more. Here are key features of Recurrent Neural Networks.

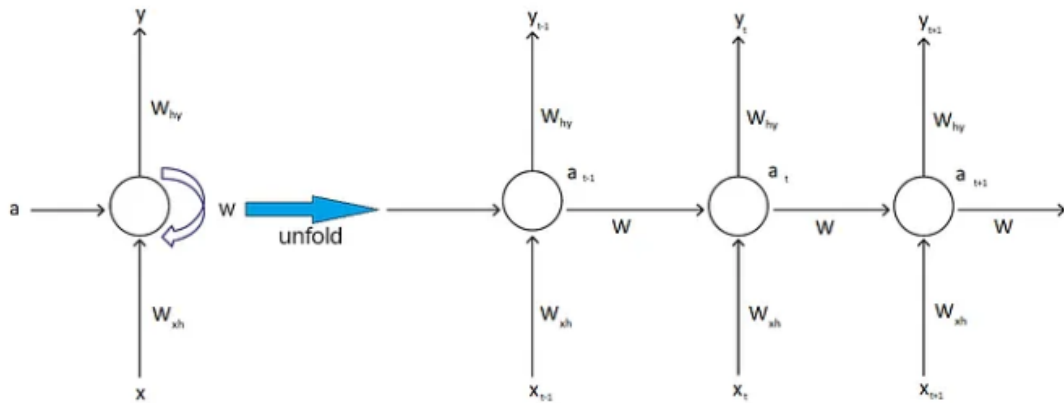


Figure 3.5: Workflow description of Recurrent Neural Network

RNNs are designed to analyze data sequences where each data point is associated with a time step. This can include time series data, text, audio, or any other sequential data. The defining characteristic of an RNN is the existence of recurrent connections, which facilitate the transfer of data between time steps. The network can keep track of prior inputs thanks to this.

RNNs maintain an internal concealed condition, shown as a vector, which serves as a memory of the past. This hidden state evolves as the network processes each time step and incorporates information from previous steps.

In an RNN, the identical weight set is applied to all time step. This weight sharing makes it possible for the RNN to discover and identify recurring motifs in sequential data.

RNNs can produce an output at each time step. For example, in a language modeling task, the network can foretell a sentence's next word at every time step.

Backpropagation through time (BPTT), a backpropagation algorithm modification designed for sequential data, is used to train RNNs.

BPTT involves calculating gradients and updating weights for each time step to reduce the discrepancy between expected and actual values, or a loss function.

The vanishing gradient problem, in which gradients get extremely small during backpropagation, can affect RNNs and make it challenging for the RNN to learn long-range dependencies. By adding gating mechanisms, advanced RNN variants like the Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) have been developed to address this problem.

3.6 Feature Extraction

Feature extraction in deep learning, particularly with pre-trained models like VGG-16, VGG-19, Inception-V3, EfficientNet-B0 and ResNet-50 involves several mathematical steps.

We load the model. VGG16 model comprises of completely connected layers after convolutional layers. The steps for this part are the architecture and weights of the model, which are predefined and loaded from a pre-trained model file.

Before feeding images into the VGG16 model, preprocessing is typically performed. While not explicitly shown in the code snippet we provided, common preprocessing steps include:

- Subtracting the mean pixel value from each channel (R, G, B).
- Scaling the pixel values by a factor (e.g., $1/255$) to bring them into the $[0, 1]$ range.

These steps help ensure that the input data is in line with the previously trained model. The mathematical steps here are simple subtraction and scaling operations, which are implemented on every pixel of the input picture.

Forward Pass Through Model is the core of feature extraction involves passing each image through the VGG16 model to obtain feature vectors (transfer values).

Convolutional layers in the VGG16 model perform mathematical convolutions between a collection of learnable filters and the image. This is one way to represent a convolutional layer's output:

$$\text{Output}(i, j, k) = \sum(\text{Filter}(k) * \text{Input}(i', j'))$$

Where:

$\text{Output}(i, j, k)$: Output feature map at position (i, j) in channel k .

$\text{Filter}(k)$: The k -th convolutional filter.

$\text{Input}(i', j')$: Input image values at position (i', j') .

Following convolutional layers, the generated feature maps are subjected to element-by-element application of activation functions such as Rectified Linear Units (ReLU):

$$\text{Activation}(i, j, k) = \max(0, \text{Output}(i, j, k))$$

Max-pooling layers down sample feature maps by taking the highest number found in each pooling window. The mathematical step for max-pooling is selecting the maximum value within a window as shown in figure 3.5.

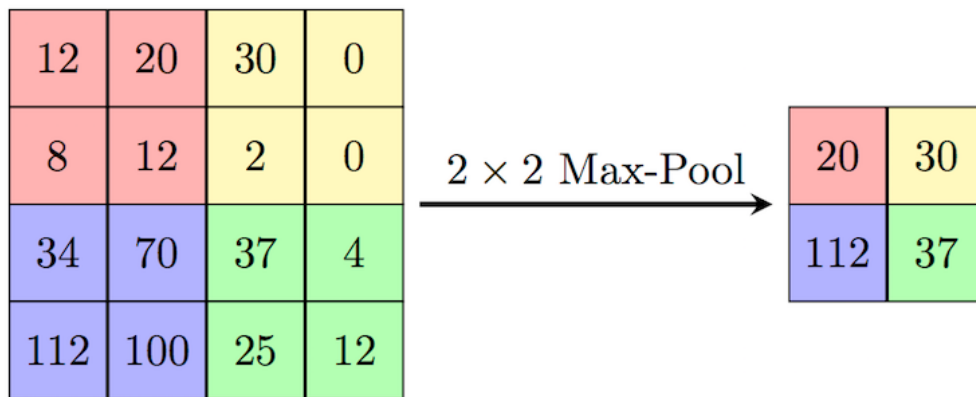


Figure 3.6: Max-pooling operation mathematical description.

The VGG16 model's last layers are completely connected layers. Mathematically, these involve matrix multiplications and activation functions like ReLU. The output of these layers is typically the feature vector that you extract.

Once the image has passed through all the layers of the VGG16 model, we obtain a feature vector. This feature vector represents high-level features of the input image and is frequently employed for a number of purposes. The mathematical steps for feature extraction involve collecting the output of the desired layer (e.g., 'fc2' layer) and using it as the feature vector.

3.6.1 Visual Geometry Group-16

The Visual Geometry Group (VGG) at the University of Oxford created the deep convolutional neural network (CNN) architecture known as the VGG16 model. In 2013, Andrew Zisserman and Karen Simonyan suggested the VGG model, and for the 2014 ImageNet Challenge, a prototype was created. They were members of VGG. It grew in popularity for image classification problems, because to its ease of use and efficiency. The "16" is the number of weight layers in VGG16. The Model Architecture Overview is as follow.

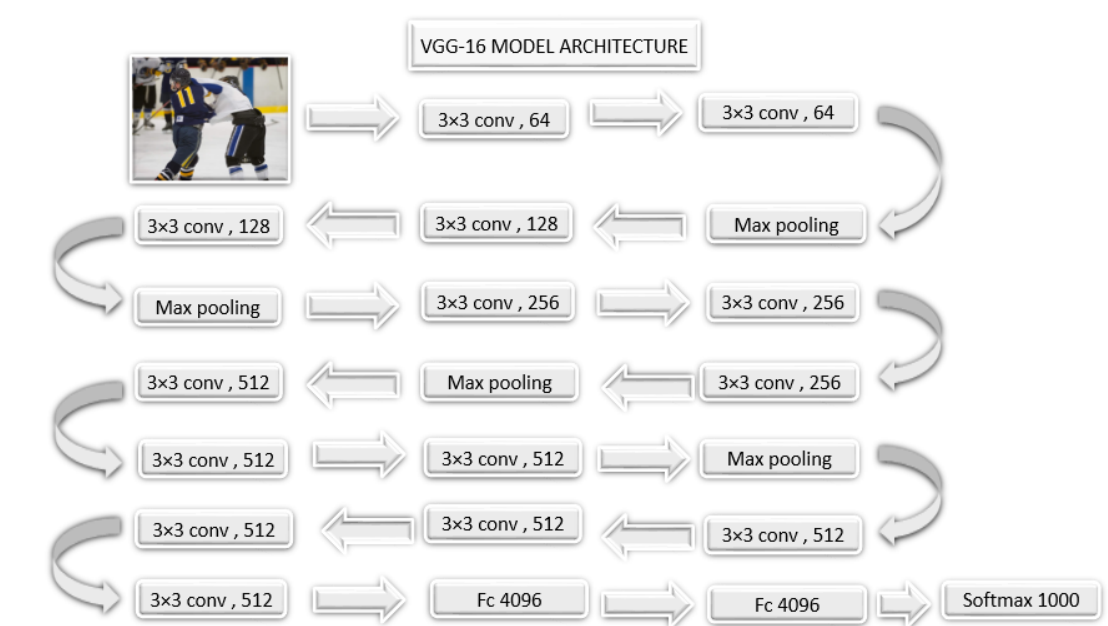


Figure 3.7: Architecture of Visual Geometry Group-16

The input to the VGG16 model is an RGB image typically of size 224x224 pixels.

The convolution operation for a single channel and a 3x3 filter at position (i, j) can be expressed as:

$$\text{Conv}(i, j) = \sum_{x=0}^2 \sum_{y=0}^2 (W(x, y) \cdot \text{Input}(i + x, j + y)) + b$$

Where:

$\text{Conv}(i, j)$ represents the output at position (i, j) in the feature map.

$W(x, y)$ are the weights of the filter.

$\text{Input}(i + x, j + y)$ denotes the input pixel values within the 3x3 region centered at (i, j) .

b represents the bias term.

The Rectified Linear Unit (ReLU) activation function is defined as:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

The max-pooling operation for a 2x2 region is expressed as:

$$\text{Max-Pool}(x) = \max(x_1, x_2, x_3, x_4)$$

Where x_1, x_2, x_3, x_4 are the values in the 2x2 region.

A fully connected layer applies a linear transformation followed by the ReLU activation function:

$$\text{FC}(x) = \text{ReLU}(W \cdot x + b)$$

Where:

$W \cdot x$ represents the matrix-vector multiplication.

b is added element-wise.

The ReLU activation function is applied to the result.

Softmax Output Layer (for Classification)

The softmax activation for class k in a classification task with K classes is defined as:

$$\text{Softmax}(z)_k = \frac{e^{z_k}}{\sum_{i=1}^K e^{z_i}}$$

Where:

z_k is the logit (raw score) for class k .

\sum represents the sum over all classes.

These mathematical equations describe the fundamental operations within the VGG16 model, where each layer applies these operations sequentially to produce the final output.

Table 3.1: VGG16 Model Architecture

Layer (type)	Output Shape
input_1 (InputLayer)	(None, 224, 224, 3)
block1_conv1 (Conv2D)	(None, 224, 224, 64)
block1_conv2 (Conv2D)	(None, 224, 224, 64)
block1_pool (MaxPooling2D)	(None, 112, 112, 64)
block2_conv1 (Conv2D)	(None, 112, 112, 128)
block2_conv2 (Conv2D)	(None, 112, 112, 128)
block2_pool (MaxPooling2D)	(None, 56, 56, 128)
block3_conv1 (Conv2D)	(None, 56, 56, 256)
block3_conv2 (Conv2D)	(None, 56, 56, 256)
block3_conv3 (Conv2D)	(None, 56, 56, 256)
block3_pool (MaxPooling2D)	(None, 28, 28, 256)
block4_conv1 (Conv2D)	(None, 28, 28, 512)
block4_conv2 (Conv2D)	(None, 28, 28, 512)
block4_conv3 (Conv2D)	(None, 28, 28, 512)
block4_pool (MaxPooling2D)	(None, 14, 14, 512)
block5_conv1 (Conv2D)	(None, 14, 14, 512)
block5_conv2 (Conv2D)	(None, 14, 14, 512)
block5_conv3 (Conv2D)	(None, 14, 14, 512)
block5_pool (MaxPooling2D)	(None, 7, 7, 512)
flatten (Flatten)	(None, 25088)
fc1 (Dense)	(None, 4096)
fc2 (Dense)	(None, 4096)
predictions (Dense)	(None, 1000)

this table shows the architecture of the VGG16 method.

3.6.2 Visual Geometry Group-19

The VGG19 model, short for "Visual Geometry Group 19," is a convolutional neural network (CNN) architecture developed by the Visual Geometry Group at the University

of Oxford. It's an extension of the VGG16 model, designed for image classification tasks. VGG19 is renowned for its ease of use and effectiveness in deep learning, and it's widely used as a benchmark in the field.

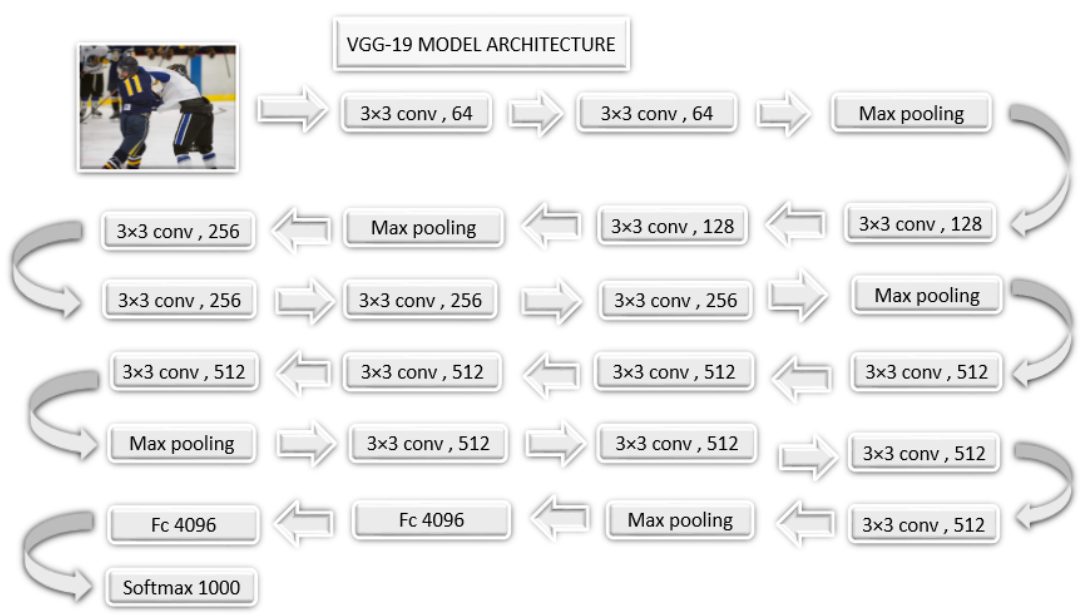


Figure 3.8: Architecture of Visual Geometry Group-19

There are 19 layers in VGG19, comprising 3 fully linked layers and 16 convolutional layers. It follows a straightforward architecture with repeated blocks of convolutional layers, with max-pooling layers in between.

The convolutional layers have small 3x3 filters, which allows the model to capture fine-grained details in images.

To provide a mathematical representation of the VGG19 model, we can break it down into its key components: completely connected layers, max-pooling layers, and convolutional layers. We'll use mathematical notation to describe these operations:

Given an input feature map X with dimensions $W_{in} \times H_{in} \times D_{in}$, where:

W_{in} is the width of the input feature map.

H_{in} is the height of the input feature map.

D_{in} is the number of input channels.

The convolutional layer applies N filters with dimensions $F \times F \times D_{\text{in}}$, where:

N is the number of filters.

F is the filter size (e.g., 3x3).

The output feature map Y is calculated as follows:

$$Y_{i,j,k} = \sum_{l=1}^{D_{\text{in}}} \sum_{m=1}^F \sum_{n=1}^F (X_{i+m-1,j+n-1,l} \cdot W_{m,n,l,k}) + b_k$$

Where:

$Y_{i,j,k}$ is the value at position (i, j, k) in the output feature map.

$X_{i+m-1,j+n-1,l}$ is the value at position $(i + m - 1, j + n - 1, l)$ in the input feature map.

$W_{m,n,l,k}$ is the weight at position (m, n, l, k) in the filter.

b_k is the bias term for the k -th filter.

Max-pooling layers reduce the spatial dimensions of the feature map by taking the maximum value in a local region. For example, a 2x2 max-pooling layer operates as follows:

$$Y_{i,j,k} = \max(X_{2i,2j,k}, X_{2i+1,2j,k}, X_{2i,2j+1,k}, X_{2i+1,2j+1,k})$$

Where:

$Y_{i,j,k}$ is the value at position (i, j, k) in the output feature map.

$X_{2i,2j,k}$ represents the input value at position $(2i, 2j, k)$, and so on.

A fully connected layer takes the result of the layer before it (flattened into a 1D vector) and performs a linear transformation, then an activation function. For simplicity, we can represent this as:

$$Y = \text{ReLU}(WX + b)$$

Where:

Y is the output vector of the fully connected layer.

X is the input vector (flattened feature map).

W is the weight matrix.

b is the bias vector.

$\text{ReLU}(x)$ represents the Rectified Linear Unit activation function.

These equations describe the mathematical operations of individual layers in the VGG19 model. The full model consists of a sequence of these layers, with each layer's output becoming the layer above's input.

While VGG19 performs well on image classification tasks, it's computationally costly because it has a lot of parameters. More recent architectures, like ResNet and Inception, offer similar or better performance with fewer parameters.

Overall, VGG19 is an influential deep learning model that has significantly influenced the computer vision area. It serves as a foundational model for understanding and building more advanced CNN architectures.

Table 3.2: VGG19 Model Architecture

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312

this table gives us the architectural representation of VGG19 model.

3.6.3 EfficientNet-B0

EfficientNet-B0 is a family of convolutional neural network architectures that are designed to be highly economical with regard to computer power while reaching cutting-edge results on a range of computer vision tasks. EfficientNet models are named with a combination of numbers and letters, such as B0, B1, B2, etc., where larger numbers indicate larger and more complex models.

Here's a general architecture of the EfficientNetB0 model.

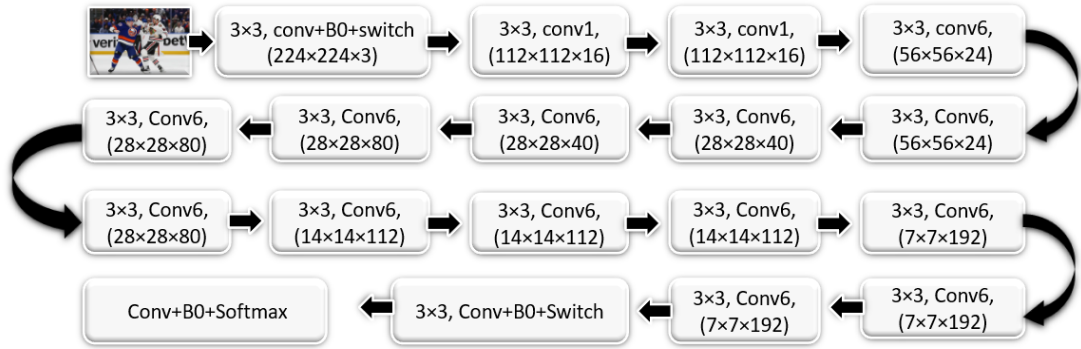


Figure 3.9: The figure shows the architecture of EfficientNetB0 model.

$$d = \alpha^\phi \quad (3.6.1)$$

$$w = \beta^\phi \quad (3.6.2)$$

$$r = \gamma^\phi \quad (3.6.3)$$

Subject to the constraint:

$$\alpha \cdot \beta^2 \cdot \gamma^2 = 2 \quad (3.6.4)$$

where $\alpha \geq 1$, $\beta \geq 1$, and $\gamma \geq 1$ 3.6.1.

According to Equation 3.6.4, FLOPS would rise by $((\alpha \cdot \beta^2 \cdot \gamma^2)\phi)$ from the original equation, where ϕ is the user-defined coefficient.

To get the predicted result, I propagate an input image x_i via the EfficientNet-B0 model. Iteratively adjusting the parameters based on the obtained gradients will train the EfficientNet-B0 model using the training subset. Measure the trained model's classification accuracy and other performance measures on the testing subset.

The input to the EfficientNetB0 model is typically an RGB image with variable size (e.g., 224x224x3 pixels or higher).

EfficientNet's overall process involves a combination of mathematical operations and design choices that optimize the compromise between performance and model size. Here's an overview of the key mathematical formulas and ideas used in the procedure:

EfficientNet models consist of multiple blocks, where each block contains a series of convolutional layers.

Pointwise convolution is performed after depthwise convolution, which applies a single convolutional filter to each input channel independently to combine the results. This is mathematically represented as:

$$Y_{i,j,k} = \sum_{l=1}^{D_{in}} \sum_{m=1}^F \sum_{n=1}^F (X_{i+m-1,j+n-1,l} \cdot W_{m,n,l,k})$$

$$Z_{i,j,k} = \sum_{l=1}^{D_{in}} Y_{i,j,l} \cdot V_{l,k}$$

Scaling Factors (Width, Depth, Resolution). Width Scaling (Width Multiplier - ϕ)

Number of Channels in Layer = Width Multiplier \times Number of Channels in Original Layer

Depth Scaling (Depth Multiplier - α)

Number of Layers in Block = Depth Multiplier \times Number of Layers in Original Block

Resolution Scaling (Resolution Multiplier - ρ)

Input Resolution = Resolution Multiplier \times Original Input Resolution

Compound Scaling Factor (S) = Width Multiplier \times Depth Multiplier

The efficient blocks involve a sequence of depthwise separable convolutions, batch normalization, and activation functions. These operations can be represented mathematically, but the details depend on the specific architecture of the block.

GAP takes the average of each feature to decrease the feature maps' spatial dimensions map:

$$Y_i = \frac{1}{\text{out} \times \text{out}} \sum_{j=1}^{\text{out}} \sum_{k=1}^{\text{out}} X_{i,j,k}$$

The final features are mapped to the output classes via the fully connected layer:

$$Y = \text{Softmax}(WX + b)$$

Where Y is the class probabilities, X is the feature vector from GAP, W is the weight matrix, and b is the bias vector.

The mathematical representation of EfficientNet's overall process involves a combination of these equations and design choices, and it can change based on the particular architecture and scaling factors used. Implementing and training an EfficientNet model typically requires deep learning frameworks that handle these mathematical details.

Table 3.3: EfficientNetB0 Model Architecture

Layer (type)	Output Shape
Input	(224, 224, 3)
Conv2D	(112, 112, 32)
MBCConv1	(112, 112, 16)
MBCConv6	(56, 56, 24)
MBCConv6	(56, 56, 24)
MBCConv6	(28, 28, 40)
MBCConv3	(28, 28, 80)
MBCConv4	(14, 14, 112)
MBCConv4	(14, 14, 112)
MBCConv4	(14, 14, 112)
MBCConv4	(14, 14, 192)
MBCConv1	(7, 7, 320)
Conv2D	(7, 7, 1280)
GlobalAvgPool	(1280,)
Fully Connected	(1000,)

this is the simplified representation of EfficientNetB0 model.

3.6.4 Residual Network-50

Convolutional neural network (CNN) architecture ResNet-50 is a member of the ResNet (Residual Network) model family. It is especially a 50-layer deep version of the original ResNet model. ResNet-50 is known for its deep architecture and it can train extremely deep networks by skipping connections effectively while mitigating the issue with fading gradients.

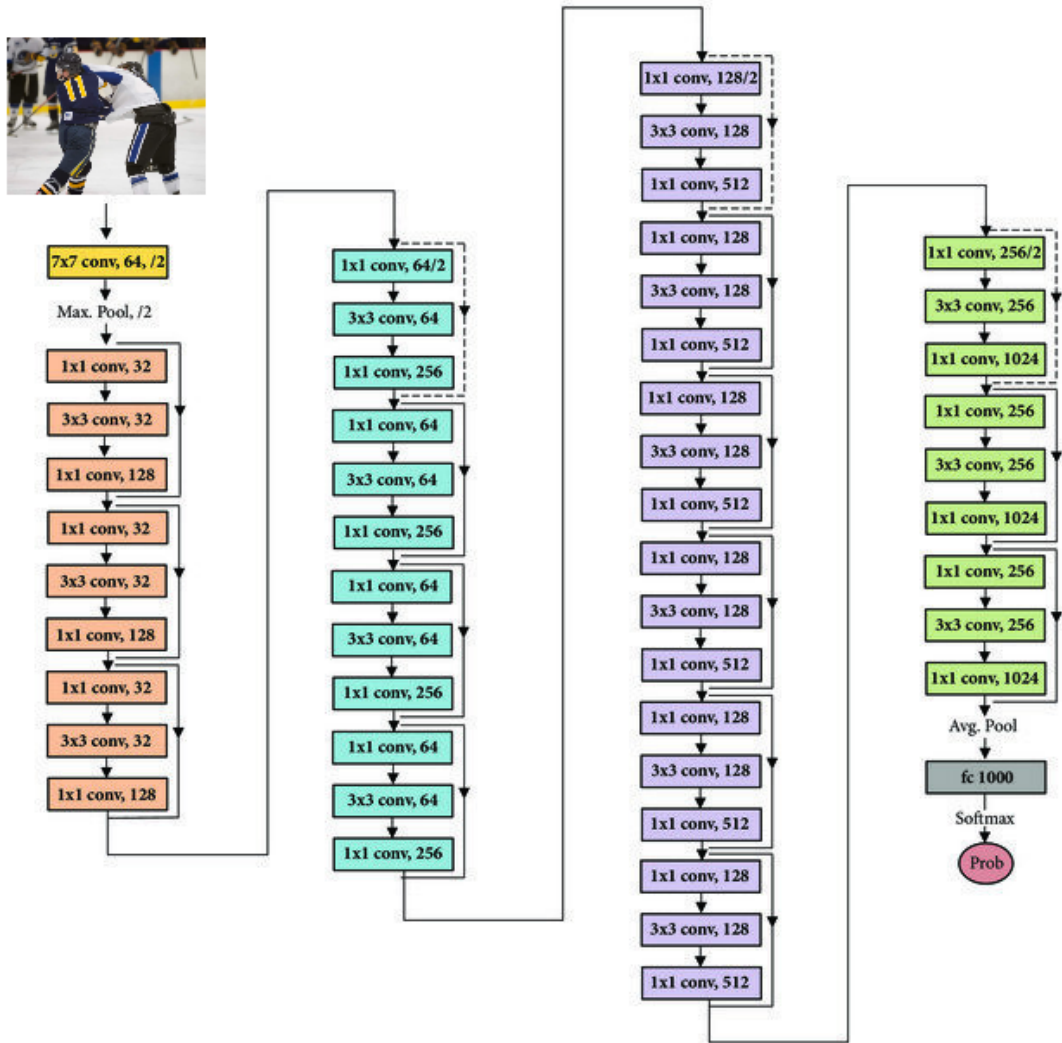


Figure 3.10: The figure shows the architecture of ResNet-50 model.

The Figure shows the architecture of resnet-50 model. The input to the ResNet-50 model is typically an RGB image with a fixed size (e.g., 224x224x3 pixels).

ResNet-50 consists of multiple convolutional layers organized into blocks. Each block contains convolutional layers in succession, batch normalization, and ReLU activation functions. The use of 3x3 convolutional filters is common in ResNet-50.

The distinctive feature of ResNet is the residual block, which is used to create deep networks. A residual block has two primary pathways: a "shortcut" or "identity" path and a "main" path. The main path contains multiple convolutional layers. The shortcut path directly connects from the source to the result of the main path, allowing gradients to flow easily during training. Mathematically, a residual block is conceivable as:

$$Y = F(X) + X$$

where X is the input and $F(X)$ represents the output of the main path.

Skip connections, also known as "identity mappings," enable the network to skip one or more layers. These connections help prevent the issue with fading gradients, making it easier to train very deep networks. In ResNet-50, skip connections are applied around each residual block.

Max-pooling layers are employed to decrease the feature maps' spatial dimensionality, typically following a series of convolutional layers. Global average pooling (GAP) is applied at the end in order to create a fixed-size feature vector and reduce spatial dimensionality.

The final fully connected layer comprising of neurons equal to the number of classes in the classification challenge makes up the network. The model's output is frequently converted using the softmax activation function into class probabilities.

ResNet-50 models are often pretrained on large image datasets like ImageNet. Pretraining helps the model learn meaningful feature representations. Transfer learning is a common approach where pretrained ResNet-50 models are fine-tuned on specific tasks.

The output layer provides class probabilities for classification tasks. For regression tasks, a different activation function and output format may be used.

ResNet-50 is a powerful and widely used CNN architecture, known for its ability to handle complex tasks in computer vision, which covers object identification, segmentation, and image classification. It has been a benchmark architecture for many image-related challenges and competitions.

Performing the entire process of training and inference with a ResNet-50 model involves a series of mathematical operations and computations. While providing a detailed mathematical representation for the entire process can be complex due to the depth and intricacy of the model, here is an overview of the key mathematical concepts and operations involved:

During forward propagation, input data (images) are passed through the network to make predictions. At each layer, the following operations are applied:

- Convolution: Convolutional input feature maps are subjected to filters in order to extract local patterns.
- Batch Normalization: Mean and variance normalization is performed to stabilize and accelerate training.
- ReLU Activation: The activation function of the Rectified Linear Unit introduces non-linearity.
- Skip Connections: Identity mappings allow gradients to flow easily, enabling training of very deep networks.
- Pooling: Max-pooling layers reduce spatial dimensions, and global average pooling computes feature vectors.

Residual blocks, represented as $Y = F(X) + X$, involve mathematical operations within each block. X represents the input to the block, and $F(X)$ represents the output of the main path, which includes convolutions, batch normalization, and ReLU. The output Y of the residual block is the element-wise sum of $F(X)$ and X . Mathematically, this is expressed as $Y_{i,j,k} = F(X)_{i,j,k} + X_{i,j,k}$.

During training, the model is optimized to minimize a loss function (e.g., cross-entropy loss for classification). Backpropagation calculates gradients with respect to model parameters (weights and biases) using the chain rule. Gradient descent or an optimizer updates model parameters to reduce the loss.

The fully connected layer applies linear transformations to the feature vectors produced by previous layers. Mathematically, $Y = WX + b$, where Y is the output, X is the input vector, W is the weight matrix, and b is the bias vector.

For classification tasks, a softmax activation function is used to convert the model's output into class probabilities. The softmax function is mathematically defined as:

$$P(Y_i = j) = \frac{e^{Y_{i,j}}}{\sum_{k=1}^C e^{Y_{i,k}}}$$

Where $P(Y_i = j)$ is the probability of the i -th example belonging to class j , $Y_{i,j}$ is the model's output for class j , and C is the number of classes.

The error between true labels and anticipated probability is quantified by the loss function. For classification, cross-entropy is a common loss function. Mathematically, the loss is computed as:

$$\text{Loss} = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C Y_{i,j} \log(P(Y_i = j))$$

Where N is the number of examples, C is the number of classes, $Y_{i,j}$ is the true label, and $P(Y_i = j)$ is the predicted probability.

This overview provides a high-level understanding of the mathematical operations involved in training and inference with a ResNet-50 model. The actual implementation and details of these operations are handled by deep learning frameworks and libraries.

Table 3.4: ResNet-50 Model Architecture (simplified form)

Layer (type)	Output Shape
input_1 (InputLayer)	(None, 224, 224, 3)
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)
conv1_conv (Conv2D)	(None, 112, 112, 64)
conv1_bn (BatchNormalization)	(None, 112, 112, 64)
conv1_relu (Activation)	(None, 112, 112, 64)
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)
conv2_block1_1_bn (BatchNormalization)	(None, 56, 56, 64)
conv2_block1_1_relu (Activation)	(None, 56, 56, 64)
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)
conv2_block1_2_bn (BatchNormalization)	(None, 56, 56, 64)
... (omitting other layers for brevity)	
conv5_block3_3_conv (Conv2D)	(None, 7, 7, 2048)
conv5_block3_3_bn (BatchNormalization)	(None, 7, 7, 2048)
conv5_block3_add (Add)	(None, 7, 7, 2048)
conv5_block3_out (Activation)	(None, 7, 7, 2048)
avg_pool (GlobalAveragePooling2D)	(None, 2048)
fc1000 (Dense)	(None, 1000)

this table give us the simplified explanation of ResNet-50 model.

3.6.5 InceptionV3

InceptionV3 is a deep convolutional neural network (CNN) architecture that is employed for picture categorization among other computer vision tasks like object detection, and image segmentation. It's a continuation of the original Inception model (GoogLeNet) and is known for its efficiency and performance.

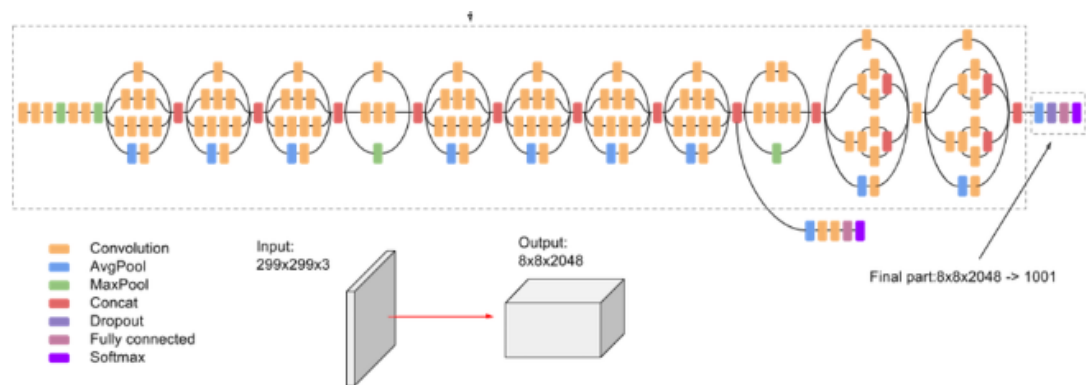


Figure 3.11: Architecture of InceptionV3

The input to the InceptionV3 model is typically an RGB image with a fixed size (e.g., 299x299 pixels).

InceptionV3 is defined by the application of "Inception modules" or "Inception blocks," which are made up of many parallel convolutional pathways. The network can now capture features at various scales thanks to these modules and complexities within a single layer. The key innovation in Inception modules is the use of multiple filter sizes (e.g., 1x1, 3x3, 5x5) in parallel and then concatenating their outputs.

In addition to the Inception modules, InceptionV3 also includes standard convolutional layers that follow each module. From the input data, these convolutional layers extract hierarchical characteristics.

Batch normalization is applied after convolutional layers to improve training stability. Activation functions (typically ReLU) give the model some non-linearity.

The feature maps' spatial dimensions are decreased through the application of max-pooling and average-pooling layers. Pooling contributes to the model's minor translation invariance and increases computational efficiency.

The final fully connected layer comprising of neurons equal to the number of classes in the classification challenge makes up the network. The model's output is frequently converted using the softmax activation function into class probabilities.

InceptionV3 models are often pretrained on large image datasets like ImageNet. Pre-training helps the model learn meaningful feature representations. Transfer learning is a common approach where pretrained InceptionV3 models are fine-tuned on specific tasks.

The output layer provides class probabilities for classification tasks. For regression tasks, a different activation function and output format may be used.

InceptionV3 is characterized by its effective utilization of computer power and its capacity to capture both local and global features effectively. It has been widely used in research and practical applications because of its impressive results on a number of computer vision tasks.

Performing the entire process of training and inference with an InceptionV3 model involves a series of mathematical operations and computations. While providing a detailed mathematical representation for the entire process can be complex due to the depth and intricacy of the model, here we give an overview of the key mathematical concepts and operations involved:

During forward propagation, input data (images) are passed through the network to make predictions. At each layer, the following operations are applied:

- Convolution: To extract local patterns from the input feature maps, convolutional filters are utilized.
- Batch Normalization: Mean and variance normalization is performed to stabilize and accelerate training.
- ReLU Activation: The activation function of the Rectified Linear Unit introduces non-linearity.
- Inception Modules: Multiple parallel convolutional pathways are combined, including 1x1, 3x3, and 5x5 convolutions.
- Pooling: Max-pooling or average-pooling layers reduce spatial dimensions.

The model is tuned to minimize a loss function during training (e.g., cross-entropy loss for classification). Backpropagation calculates gradients in relation to the model

parameters (weights and biases) using the chain rule. Gradient descent or an optimizer updates model parameters to reduce the loss.

The fully connected layer applies linear transformations to the feature vectors produced by previous layers. Mathematically,

$$Y = WX + b$$

where Y is the output, X is the input vector, W is the weight matrix, and b is the bias vector.

For classification tasks, a softmax activation function is used to convert the model's output into class probabilities. The softmax function is mathematically defined as:

$$P(Y_i = j) = \frac{e^{Y_{i,j}}}{\sum_{k=1}^C e^{Y_{i,k}}}$$

where $P(Y_i = j)$ is the probability of the i -th example belonging to class j , $Y_{i,j}$ is the model's output for class j , and C is the number of classes.

The error between true labels and anticipated probability is quantified by the loss function. Typical loss functions include cross-entropy for classification. Mathematically, the loss is computed as:

$$\text{Loss} = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C Y_{i,j} \log(P(Y_i = j))$$

where N is the number of examples, C is the number of classes, $Y_{i,j}$ is the true label, and $P(Y_i = j)$ is the predicted probability.

This overview provides a high-level understanding of the mathematical operations involved in training and inference with an InceptionV3 model. The actual implementation and details of these operations are handled by deep learning frameworks and libraries.

Table 3.5: InceptionV3 Model Architecture

Layer (type)	Output Shape
Input (InputLayer)	(None, 299, 299, 3)
Conv2d_1a_3x3 (Conv2D)	(None, 149, 149, 32)
BatchNormalization	(None, 149, 149, 32)
Activation	(None, 149, 149, 32)
Conv2d_2a_3x3 (Conv2D)	(None, 147, 147, 32)
BatchNormalization	(None, 147, 147, 32)
Activation	(None, 147, 147, 32)
MaxPooling2d_3a_3x3 (MaxPooling2D)	(None, 73, 73, 32)
Conv2d_3b_1x1 (Conv2D)	(None, 73, 73, 80)
BatchNormalization	(None, 73, 73, 80)
Activation	(None, 73, 73, 80)
Conv2d_4a_3x3 (Conv2D)	(None, 71, 71, 192)
BatchNormalization	(None, 71, 71, 192)
Activation	(None, 71, 71, 192)
MaxPooling2d_5a_3x3 (MaxPooling2D)	(None, 35, 35, 192)
...	...
Mixed_7b (Concatenate)	(None, 17, 17, 768)
...	...
AveragePooling2D (GlobalAveragePooling2D)	(None, 2048)
Dropout	(None, 2048)
Dense (Dense)	(None, 1000)

3.7 Classification Method

The classification methods are Long and Short Term Memory (LSTM) and Gradient Recurrent Unit (GRU)

3.7.1 Long and Short Term Memory (LSTM)

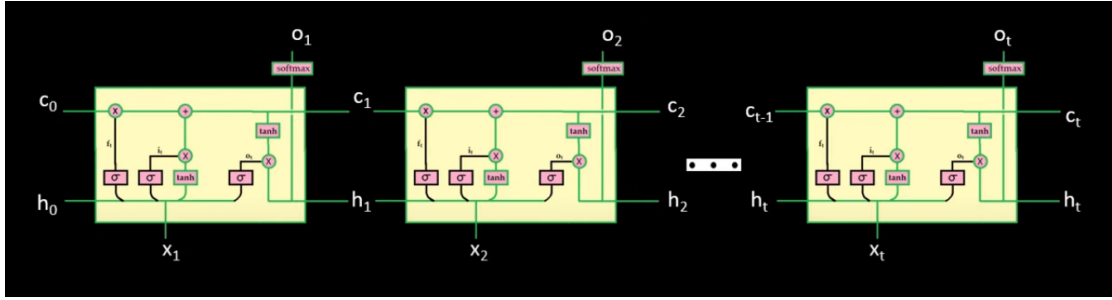


Figure 3.12: Systematic diagram of Long and Short Term Memory (LSTM)

An LSTM cell consists of several mathematical components, including gates, input transformations, and cell state updates:

Input Gate (i_t):

Computes the amount by which the The cell state has to be updated with fresh data. Mathematically, the input gate is computed using the sigmoid activation function:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

where W_i and b_i are weights and biases, h_{t-1} is the previous hidden state, and x_t is the current input.

Forget Gate (f_t):

Computes the amount by which the previous cell state should be forgotten. Mathematically, the forget gate is computed using the sigmoid activation function:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Cell State Update (\tilde{C}_t):

Computes the new candidate cell state. Mathematically, the candidate cell state is computed using the hyperbolic tangent activation function:

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

Output Gate (o_t):

Computes the amount by which the cell state should be exposed in the output. The output gate is calculated mathematically using the sigmoid activation function:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

New Cell State (C_t):

Computes the updated cell state using the input gate, forget gate, and candidate cell state:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Hidden State (h_t):

Computes the updated hidden state using the output gate and the updated cell state:

$$h_t = o_t \cdot \tanh(C_t)$$

The LSTM cell's mathematical equations describe how information is processed and propagated through the cell. In a full LSTM network, you stack multiple LSTM cells and potentially add additional layers for more complex architectures. The architecture design involves configuring the quantity of units, layers, and LSTM cells in each layer.

Loss Function (L):

Choose an appropriate loss function L based on the nature of your sequence prediction task. For example, for sequence classification, you might use categorical cross-entropy:

$$L(Y \cdot \hat{Y}) = - \sum_{t=1}^T \sum_{c=1}^C Y_{t,c} \cdot \log(Y^{t,c})$$

Here, T is the length of the sequence, C is the number of classes, $Y_{t,c}$ is the true label, and $Y^{t,c}$ is the predicted probability.

Optimization Algorithm:

Choose an optimization algorithm (e.g., Adam, RMSProp, SGD) to update the LSTM model's parameters during training. Define the learning rate (α) and other hyperparameters of the chosen optimizer.

Model Compilation:

Compile the LSTM model by specifying the chosen loss function and optimizer:

```
model.compile(loss = L, optimizer = optimizer( $\alpha$ ))
```

Mathematically, this step involves selecting the appropriate loss function for your sequence prediction task, which measures the discrepancy between actual and predicted sequences. Additionally, you choose a technique for optimization utilizing gradient descent to update the LSTM parameters of the model in a manner that reduces the chosen

loss function. The optimizer’s hyperparameters, such as learning rate, influence the rate of parameter updates during training.

Given input sequences \mathbf{X} and corresponding target sequences \mathbf{Y} , let’s outline the mathematical steps involved in training an LSTM model.

Forward Propagation the input sequence \mathbf{X} and the model’s initial parameters, perform forward propagation to compute the predicted output sequence $\hat{\mathbf{Y}}$ for each time step t .

Calculate the loss $L(\hat{\mathbf{Y}}, \mathbf{Y})$ between the predicted output sequence $\hat{\mathbf{Y}}$ and the true target sequence \mathbf{Y} using the chosen loss function.

Backpropagation Through Time (BPTT) compute’s the gradients of the loss across time using backpropagation with regard to the model’s parameters. BPTT includes figuring out the gradients for every time step t and accumulating them over the entire sequence.

Model parameter updating through the use of an optimization algorithm (e.g., gradient descent). For each parameter w and its corresponding gradient $\frac{\partial L}{\partial w}$, update the parameter using the learning rate α : $w \leftarrow w - \alpha \frac{\partial w}{\partial L}$

Iterate over multiple epochs, repeating the forward and backward propagation steps to gradually raise the model’s performance. Mathematically, training involves a series of calculations, including matrix multiplications, activation functions (e.g., sigmoid, hyperbolic tangent), loss calculations, and gradient computations. The goal is to optimize the model’s parameters so that it can produce accurate sequence predictions while minimizing loss.

For the trained LSTM model and a validation set with input sequences \mathbf{valX} and target sequences \mathbf{valY} , let’s outline the mathematical steps involved in evaluating the model:

For each input sequence \mathbf{valX} , perform forward propagation using the trained LSTM model to compute the predicted output sequence $\hat{\mathbf{valY}}$ for each time step t .

Calculate the loss $L(\hat{\mathbf{valY}}, \mathbf{valY})$ between the predicted output sequence $\hat{\mathbf{valY}}$ and the true target sequence \mathbf{valY} using the same chosen loss function used during training.

Compute evaluation metrics based on the predicted output sequence $\hat{\mathbf{valY}}$ and the true target sequence \mathbf{valY} . Metrics can include loss, accuracy, or custom metrics specific to the sequence prediction task. Mathematically, this step involves using the trained model to make predictions on the validation set and then calculating the loss and other evaluation metrics based on the predicted sequences and true target sequences. The cal-

culations involve comparing predicted sequences with actual sequences and aggregating the results to compute metrics that assess the model’s performance.

Now let’s consider the mathematical steps involved in hyperparameter tuning:

Define a set of hyperparameters that need to be adjusted, like the quantity of LSTM layers, units per layer, learning rate, batch size, etc. Specify possible values or ranges for each hyperparameter.

Choose a hyperparameter tuning method, such as grid search or random search. **Grid Search:** Systematically iterate through every conceivable pairing of hyperparameters inside the specified search space. Take a random sample of hyperparameter combinations from the search space. **Iterate Over Hyperparameter Combinations:**

For each combination of hyperparameters \mathbf{H} : Utilizing the provided hyperparameters and training data, train the LSTM model.

Utilizing the validation data, assess the LSTM model’s performance using the chosen evaluation metrics. Calculate metrics such as loss, accuracy, or other relevant metrics based on the predicted sequences and true targets.

Choose the combination of hyperparameters that leads to the best performance on the validation data. This is often the combination that yields the lowest loss or the highest accuracy, depending on the task. Mathematically, hyperparameter tuning involves systematic iteration and evaluation. While the mathematical operations themselves may not be complex, the process relies on training and evaluating the LSTM model with different hyperparameter combinations to identify the settings that optimize performance on unseen data.

Given a trained LSTM model and a set of input sequences $\mathbf{new_X}$, let’s outline the mathematical steps involved in making predictions:

Forward Propagation for Prediction: For each input sequence $\mathbf{new_X}$, perform forward propagation using the trained LSTM model to compute the predicted output sequence $\mathbf{new_Y}$ for each time step t . Mathematically, this step involves using the trained LSTM model to propagate the input sequences $\mathbf{new_X}$ through the network, resulting in predicted output sequences $\mathbf{new_Y}$. The calculations involved include matrix multiplications, activation functions, and element-wise operations within the LSTM layers to generate the predicted output sequences.

Evaluate the LSTM model’s performance on validation and test data using chosen metrics.

Identify areas where the model is underperforming or not meeting your expectations.

Formulate hypotheses about what changes could potentially improve the model’s performance.

Make changes to the model architecture, hyperparameters, or data preprocessing based on your hypotheses.

Evaluate the model with the changes on validation and test data and compare the results with previous iterations.

If the changes lead to improvements, iterate further by making additional adjustments.

Ensure that any improvements generalize well to the test data and that you’re not overfitting to the validation set. While the mathematical aspect of this step involves experimentation, analysis, and comparison of results, it doesn’t involve specific mathematical equations. Instead, it’s about using data-driven insights and domain knowledge to refine the model iteratively and make it more effective in its task.

Table 3.6: LSTM Model Architecture

Layer	Description
Input Layer	Input sequence of vectors
LSTM Layer	Long Short-Term Memory (LSTM) cells
	Cell State (C_t) Hidden State (h_t)
Output Layer	Final output layer

3.7.2 Gated Recurrent Unit (GRU)

One kind of recurrent neural network (RNN) architecture called the Gated Recurrent Unit (GRU) was created to overcome some of the drawbacks of conventional RNNs, like the vanishing gradient issue. GRUs have gating mechanisms that allow them to capture long-range dependencies in sequential data while mitigating some of the training challenges associated with standard RNNs.

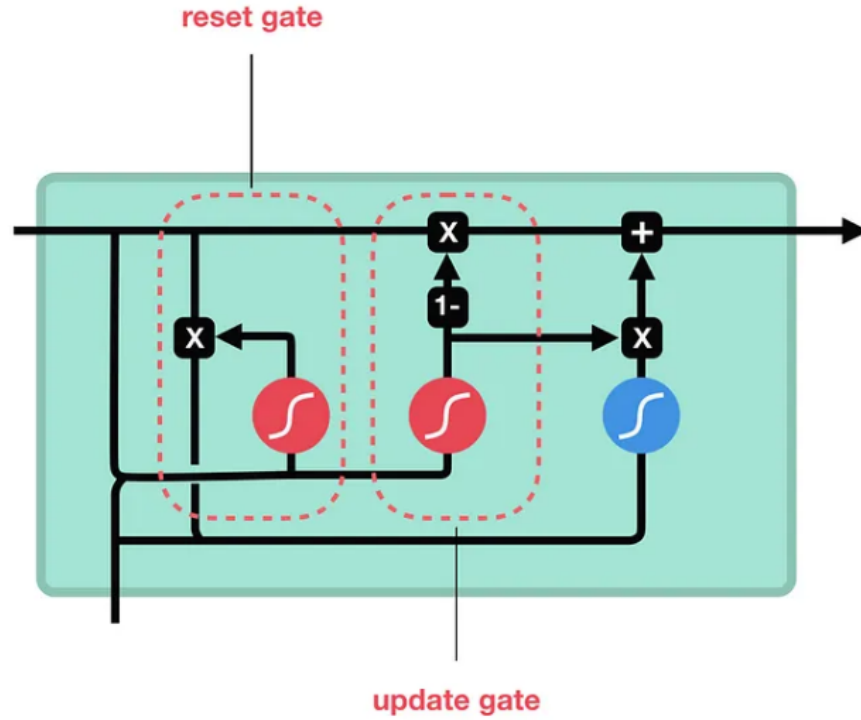


Figure 3.13: Systematic representation of Gated Recurrent Unit.

Working Principles of GRU is as follow

- **Hidden State:** Like traditional RNNs, GRUs maintain a hidden state vector h_t at each time step t . Information regarding the sequential context thus far observed is captured by this hidden state.
- **Gating Mechanisms:** GRUs have two gating mechanisms:
 - **Reset Gate (r_t):** chooses which data from the earlier concealed state h_{t-1} should be forgotten or reset.
 - **Update Gate (z_t):** Regulates the amount of the new candidate hidden state \tilde{h}_t should be included in the updated hidden state h_t .
- **Candidate Hidden State (\tilde{h}_t):** A candidate hidden state \tilde{h}_t is computed at each time step based on the input x_t and the reset gate r_t . It captures new information from the current input.
- **Updating the Hidden State:** Update gate z_t determines what proportion of the new candidate hidden state \tilde{h}_t should be mixed with the previous hidden state

h_{t-1} . The updated hidden state h_t is then computed as a weighted combination of \tilde{h}_t and h_{t-1} , where the update gate z_t controls the weighting.

The mathematical equations for a GRU are as follows:

$$\text{Reset Gate: } r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\text{Update Gate: } z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$\text{Candidate Hidden State: } \tilde{h}_t = \tanh(W_h \cdot [r_t \cdot h_{t-1}, x_t])$$

$$\text{Updated Hidden State: } h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t$$

In these equations, x_t represents the input at time step t , h_t is the hidden state at time step t , and W_r , W_z , and W_h are weight matrices learned during training.

GRUs are capable of capturing long-rangerelationships in sequential data, which makes them appropriate for tasks like time series prediction, speech recognition, and natural language processing. They have become popular alternatives to traditional RNNs and are often used in modern deep learning architectures.

Table 3.7: GRU Model Architecture

Layer	Description
Input Layer	Input sequence of vectors
GRU Layer	Gated Recurrent Unit (GRU) cells
	Reset Gate (r_t) controls forgetting Update Gate (z_t) controls updating Current Memory (\tilde{h}_t) candidate memory Previous Memory (h_{t-1}) previous memory Output (h_t) updated memory
Output Layer	Final output layer

3.8 Activation Functions

We have used 3 activation fuction for our models sigmoid activation function, ReLU activation function and softmax activation.

3.8.1 Rectified Linear Unit (ReLU)

In artificial neural networks, the Rectified Linear Unit (ReLU) activation function is frequently utilized, particularly in deep learning models. It introduces model's non-linearity and has several advantages, including alleviating the vanishing gradient problem and computational efficiency.

The ReLU function is defined as follows:

$$\text{ReLU}(x) = \max(0, x)$$

Where:

- $\text{ReLU}(x)$ is the output of the ReLU activation for input x . - x is the input value.

Key properties and characteristics of the ReLU activation function:

- Rectification: The ReLU function returns the input x if x is either larger than or equal to zero; if not, zero is returned. In other words, it "rectifies" negative values to zero, maintaining the positive values unaltered. - Non-linearity: ReLU gives the model non-linearity, which enables neural networks to learn intricate, non-linear relationships in data. - Sparsity: ReLU activations are sparse, meaning they can make some neurons inactive (outputting zero) during training, which can lead to efficient model training and reduced overfitting. - Computational Efficiency: ReLU involves only a basic comparison and maximum operation, making it computationally efficient to compute.

While ReLU has many advantages, it's worth noting that it's not without its limitations. The "dying ReLU" problem, in which neurons might become trapped, is one of ReLU's problems in an inactive state (always outputting zero) during training. This can happen when the weights associated with a neuron are updated in a way that makes the neuron's output always negative. To address this problem, variants of ReLU, such as Leaky ReLU, Parametric ReLU (PReLU), and Exponential Linear Unit (ELU), have been proposed, which allow a small gradient for negative inputs to prevent neurons from dying.

3.8.2 Sigmoid Activation Function:

The logistic function, sometimes known as the sigmoid activation function, is a common activation function used in artificial neural networks and logistic regression models. It

makes it appropriate for binary classification problems by mapping input values to a range between 0 and 1 where the goal is to produce likelihoods that a given input falls into a specific class.

The sigmoid function is defined as follows:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Where:

- $\sigma(z)$ is the sigmoid function's result for a particular input z . - e is the natural logarithm's base (approximately equal to 2.71828).

Key properties of the sigmoid function:

- Range: Sigmoid function's output is always in the range of 0 and 1, therefore when used for binary classification, it can give output can be interpreted as a probability. - S-shaped Curve: The sigmoid function has an "S" shape, which means that minor adjustments to the input value result in relatively minor adjustments to the output. This makes it suitable for gradient-based optimization algorithms. - Smooth and Continuous: The sigmoid function is smooth and differentiable everywhere, which allows for gradient-based training methods like backpropagation. - Centered at 0.5: The sigmoid function has a midpoint at $z = 0$, where $\sigma(0) = 0.5$. This means that when the input is 0, the sigmoid function produces an output of 0.5, which can be understood as an equal likelihood of being a member of either class in a binary classification problem.

3.8.3 Softmax Activation Function:

In the output layer of neural networks for multi-class classification issues, the softmax activation function is a frequently employed activation function. It converts logits or raw scores into probability distributions for a number of classes, making it suitable for problems where an input can belong to one of several mutually exclusive classes.

The softmax function is defined as follows for a vector of raw scores or logits \mathbf{z} :

$$\text{Softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Where:

- $\text{Softmax}(\mathbf{z})_i$ is the output of the softmax function for class i . - z_i is the raw score or logit associated with class i . - K is the total number of classes.

Key properties and characteristics of the softmax activation function:

- Normalization: The softmax function normalizes the raw scores, ensuring that the output values are non-negative and sum to 1. This property allows the softmax function to represent the probabilities of each class. - Probability Interpretation: The likelihood that the input belongs to a given class can be deduced from the output of the softmax function for each class. The predicted class is the one with the highest likelihood. - Mutually Exclusive: The softmax function assumes that each input belongs to exactly one of the classes (mutually exclusive). - Smoothness: The softmax function is smooth and differentiable everywhere, making it suitable for gradient-based optimization algorithms like backpropagation.

The softmax activation is often utilized in a neural network's last layer for tasks like image classification, text categorization, and various multi-class classification problems. It transforms the raw scores produced by the preceding layers into class probabilities, allowing the model to give probabilistic forecasts regarding the input's class membership.

3.9 Model Validation

We start by splitting your dataset into two subsets: the training set and the test set. The size of our training set (N_{train}) is calculated as a percentage of the total dataset size. In our code, we used 80 percent of the data for training ($N_{\text{train}} = \text{int}(\text{len}(\text{names}) \times 0.8)$), leaving 20 percent for testing ($N_{\text{test}} = \text{int}(\text{len}(\text{names}) \times 0.2)$). Mathematically, we can represent the dataset split as follows: Training Set:

Size: N_{train}

Data: X_{train}

Labels: Y_{train}

Test Set:

Size: N_{test}

Data: X_{test}

Labels: Y_{test}

we specify hyperparameters for training:

- Number of epochs (E): In our code, we set $E = 200$.
- Batch size (B): our batch size is $B = 500$.

we use the training data $(X_{\text{train}}, Y_{\text{train}})$ to train the model. The training process entails adjusting the model's weights repeatedly in order to minimize a loss function (L) using an optimization algorithm. Mathematically, this process can be represented as:

Model: $M(\theta)$ (with trainable parameters θ)

Loss: $L(Y_{\text{train}}, M(X_{\text{train}}, \theta))$

Optimization: $\theta \leftarrow \theta - \alpha \nabla L(Y_{\text{train}}, M(X_{\text{train}}, \theta))$

where α is the learning rate.

After training, we assess the model's performance utilizing the test dataset $(X_{\text{test}}, Y_{\text{test}})$. The evaluation computes various metrics, including loss (L_{test}) and accuracy (A_{test}). Mathematically, model evaluation can be expressed as:

Loss on Test Set: $L_{\text{test}} = L(Y_{\text{test}}, M(X_{\text{test}}, \theta))$

Accuracy on Test Set: $A_{\text{test}} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} I(M(X_{\text{test}}^{(i)}, \theta) = Y_{\text{test}}^{(i)})$

where $I(\text{condition})$ is a function that indicates whether a condition is true by equaling 1 and 0 otherwise.

In summary, the model validation process involves mathematically defining datasets, training the model with hyperparameters, updating model weights in order to reduce the loss, and evaluating the model's performance on the test dataset using metrics like loss and accuracy. This process helps evaluate the model's ability to generalize to new data.

3.10 Performance Evaluation

For performance evaluation we have accuracy, sensitivity, specificity, precision, f1-score, recall and loss.

	Actual Positive (1)	Actual Negative (0)
Predicted Positive	TP	FP
Predicted Negative	FN	TN

From the values in the confusion matrix, we can compute various performance metrics as follows:

Accuracy:

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

The accuracy formula computes the percentage of correctly identified cases (including true positives and true negatives) in the dataset.

Sensitivity (True Positive Rate or Recall):

Sensitivity assesses The capacity of the model to accurately detect positive cases (class 1). It is computed in this way:

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

Sensitivity tells us how well our model identifies the actual positive cases. A high sensitivity demonstrates how well the model can identify positives.

Specificity (True Negative Rate):

Specificity assesses the model's ability to correctly detect negative cases. (class 0). It is computed in this way:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Specificity tells us how well our model identifies the actual negative cases. A high specificity suggests that the model is effective at minimizing false negative alarms.

Precision:

Precision evaluates the model's accuracy in making favorable predictions. It is computed in this way:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision indicates how many of the projected positive cases were found to be positive.

A high precision indicates that the positive predictions are reliable.

F1 Score:

The F1 score is a harmonic mean of precision and memory that offers a balance. It is computed in this way:

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

When we wish to account false negatives as well as false positives, we can utilize the F1 score.

Recall (Same as Sensitivity):

Recall, also known as Sensitivity, assesses the model's ability to properly detect positive events (class 1). It is computed in this way:

$$\text{Recall} = \frac{TP}{TP + FN}$$

CHAPTER 4

Results

This section contains the explanation of our finding which we have obtained by using the feature extraction methods and classification methods. We have used 5 feature extraction methods. The feature extraction methods are Visual Geometry Group-16, Visual Geometry Group-19, Inception-V3, ResNet-50 and EfficientNetB0. The classification methods are Long and Short Term Memory (LSTM) and Gradient Recurrent Unit (GRU).

The method consists of taking a selected group of video frames, sending them to the pre-trained network, getting the results from one of its final layers, and then training a new network architecture using the LSTM or GRU special-purpose neurons from these outputs. These neurons contain memories and are able to analyze the temporal information in the videos if they spot violence at any point, the video will be labelled as violent. First we have imported some of the important models, layers and applications which are helpful in running our code. Then after that we have checked our keras (It is a high-level neural network library) version. After that we have define a function in which we are using a helper function print-progress to print the amount of videos processed the datasets. Then in the next step we have loaded our data by defining the directory to place the video dataset. Then after that we have defined some of the data dimensions of our convenience. Then define a function that is used to get 20 frames from a video file and convert the frame to a suitable format for the neural net. Then we have defined another helper function to get the names of the data downloaded and label it. Then we have plotted a video frame to see if the data is correct. Then we are going to load 20 frames per video. Convert back the frames to uint8 pixel format to plot the frame. After that we will apply our pre-trained model and see its summary. We can observe the anticipated

tensor shapes used as input by the trained model. In this instance, it is $224 \times 224 \times 3$ image. Note that we have defined the frame size as $224 \times 224 \times 3$ but it varies from method to method. The video frame will be the input of the image mode. Now then classification layer that we going to use is the one that is fully connected and dense. After that, we defined a function to process video frames through our pre-trained model and get the transfer values. We save the so-called Transfer Values to a cache file shortly before the pre-trained model's last classification layer. The reason for employing a cache-file is that processing a picture with the pre-trained model takes a lengthy time. We can save if each image is processed more than once, a great deal of time can be saved by storing the transfer values. After running all of the movies through the pre-trained transfer-values from the model and storing them in a cache file, we use those transfer-values as input to our LSTM or GRU neural structure. Next, the classes from the neural network will be used to train the violence dataset (Violence, No-Violence), so that the network may learn how the transfer-values from the pre-trained model are used to classify photos. We define a functions to get the transfer values from pre-trained model with defined number of files training and testing. We are going to split the data set into training set and testing set. The training set is used to train the model and the test set to check the model accuracy. Then we are going to process all video frames through feature extraction methods and saving the transfer values. After that we have made files for training set and test set. We have already saved all the videos transfer values into disk. But we have to load those transfer values into memory in order to train the LSTM or GRU net. One question would be why not process transfer values and load them into RAM memory? Yes is a more efficient way to train the second net. But if you have to train the LSTM or GRU in different ways in order to see which way gets the best accuracy, if you didn't save the transfer values into disk you would have to process the whole video each training. It's very time-consuming to process the videos through a pre-trained model net. In order to load the saved transfer values into RAM memory we are going to use two functions. The purposed models are EfficientNETB0-LSTM, Resnet-LSTM, VGG16-LSTM, VGG19-LSTM, InceptionV3-LSTM, EfficientNETB0-GRU, Resnet50-GRU, VGG16-GRU, VGG19-GRU and InceptionV3-GRU. The performance of these methods are explained in Table 4.1.

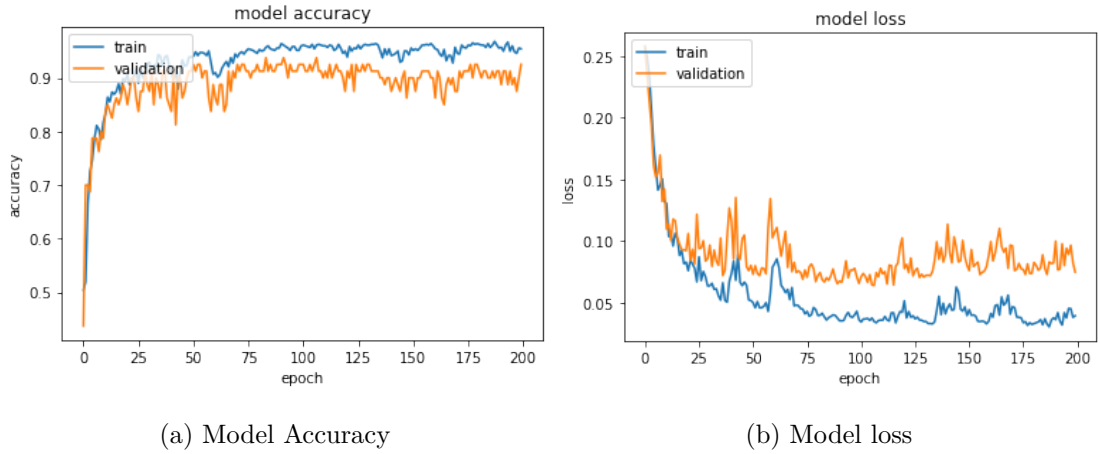


Figure 4.1: InceptionV3-LSTM method graphical representation of loss and accuracy.

The graphs displays the accuracy and loss of the Test and Train datasets while using the fundamental InceptionV3-LSTM model. The y-axis shows the accuracy and loss and the x-axis shows the model training epochs . In this we have attained 91 percent accuracy and 7.96 percent loss. These results we have attained by first time running the model.

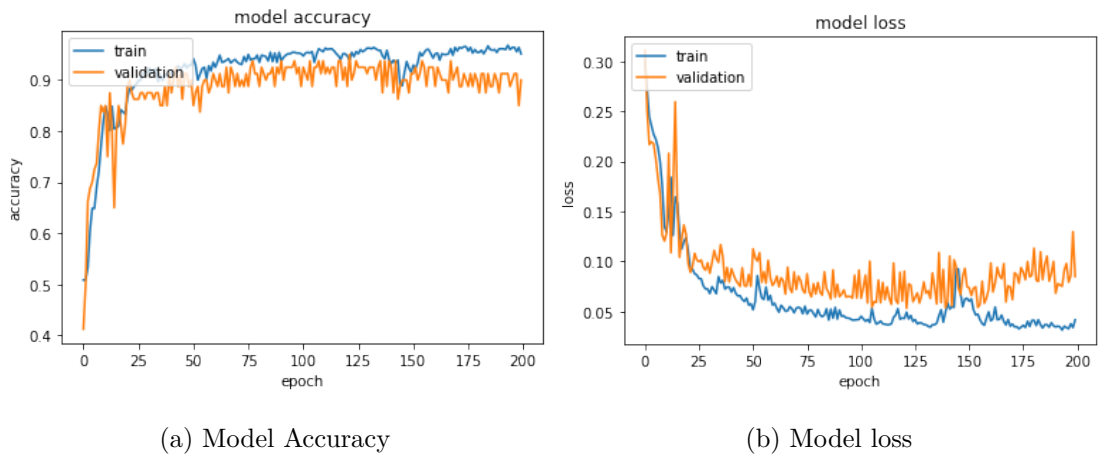


Figure 4.2: InceptionV3-GRU method graphical representation of loss and accuracy.

The graphs displays the accuracy and loss of the Test and Train datasets while using the fundamental Inception V3-GRU model. The y-axis shows the accuracy and loss and the x-axis shows the model training epochs. In this we have attained 89.5 percent accuracy and 8.42 percent loss. These results we have attained by first time running the model.

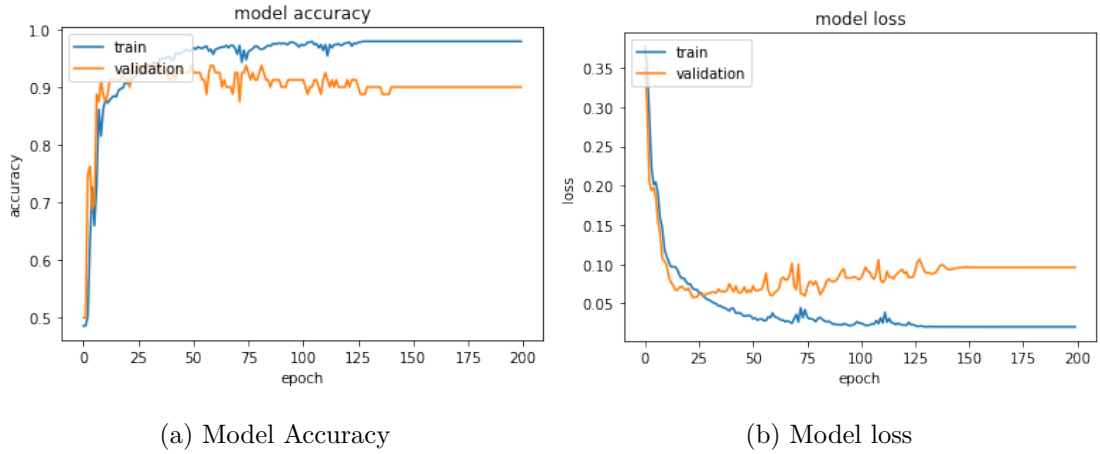


Figure 4.3: EfficientNetB0-GRU method graphical representation of loss and accuracy.

The graphs displays the accuracy and loss of the Test and Train datasets while using the fundamental EfficientNetB0-GRU model. The y-axis shows the accuracy and loss and the x-axis shows the model training epochs. In this we have attained 92 percent accuracy and 7.74 percent loss. These results we have attained by first time running the model.

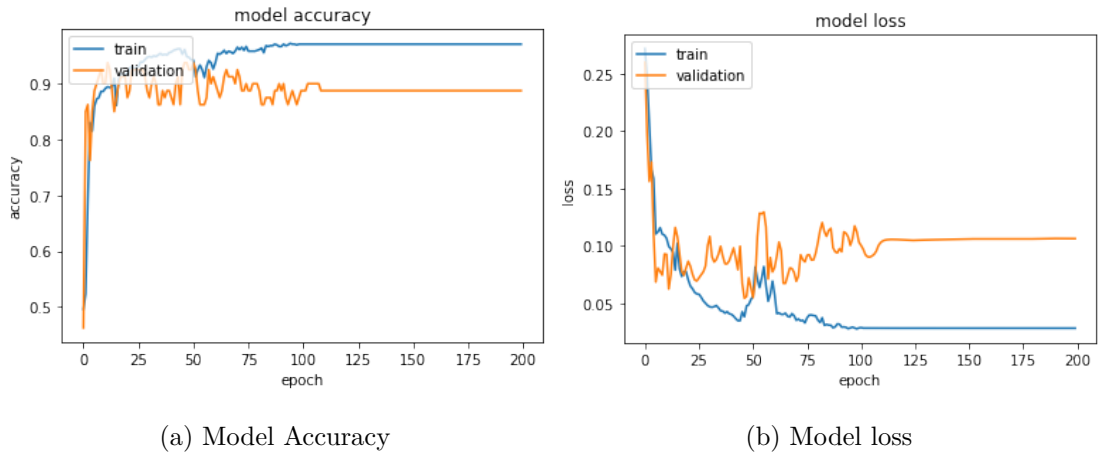


Figure 4.4: EfficientNetB0-LSTM method graphical representation of loss and accuracy.

The graphs displays the accuracy and loss of the Test and Train datasets while using the fundamental EfficientNetB0-LSTM model. The y-axis shows the accuracy and loss and the x-axis shows the model training epochs. In this we have attained 89 percent accuracy and 10.28 percent loss. These results we have attained by first time running the model.

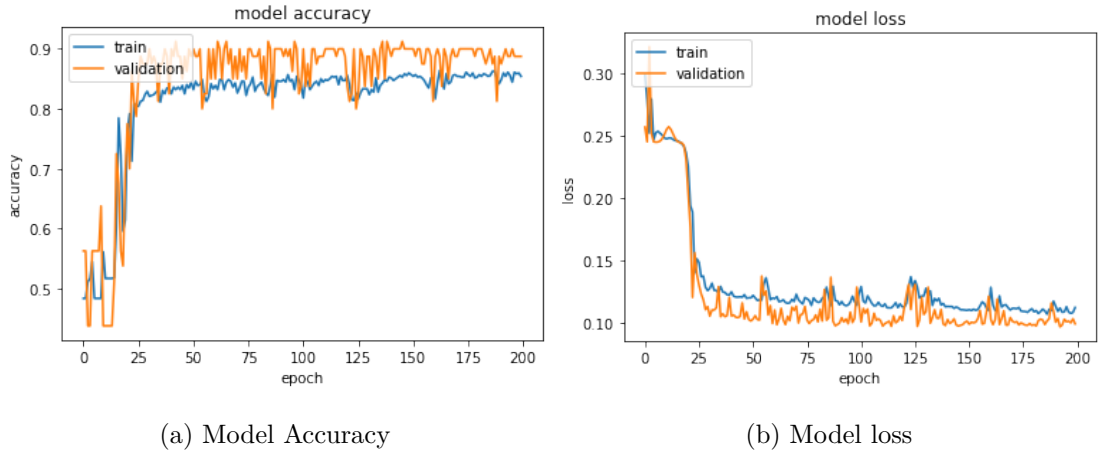


Figure 4.5: Resnet50-GRU method graphical representation of loss and accuracy.

The graphs displays the accuracy and loss of the Test and Train datasets while using the fundamental Resnet50-GRU model. The y-axis shows the accuracy and loss and the x-axis shows the model training epochs. In this we have attained 86.5 percent accuracy and 11.29 percent loss. These results we have attained by first time running the model.

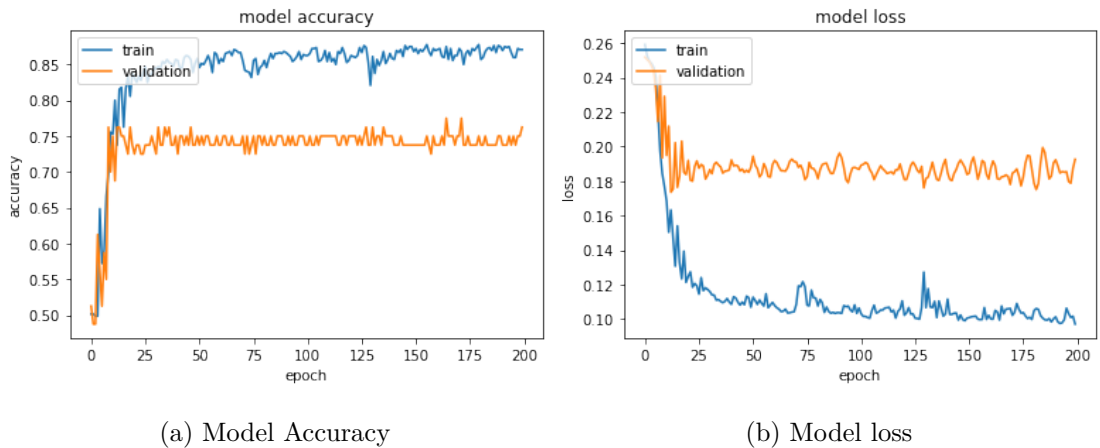


Figure 4.6: Resnet50-LSTM method graphical representation of loss and accuracy.

The graphs displays the accuracy and loss of the Test and Train datasets while using the fundamental Resnet50-LSTM model. The y-axis shows the accuracy and loss and the x-axis shows the model training epochs. In this we have attained 81.5 percent accuracy and 13.41 percent loss. These results we have attained by first time running the model.

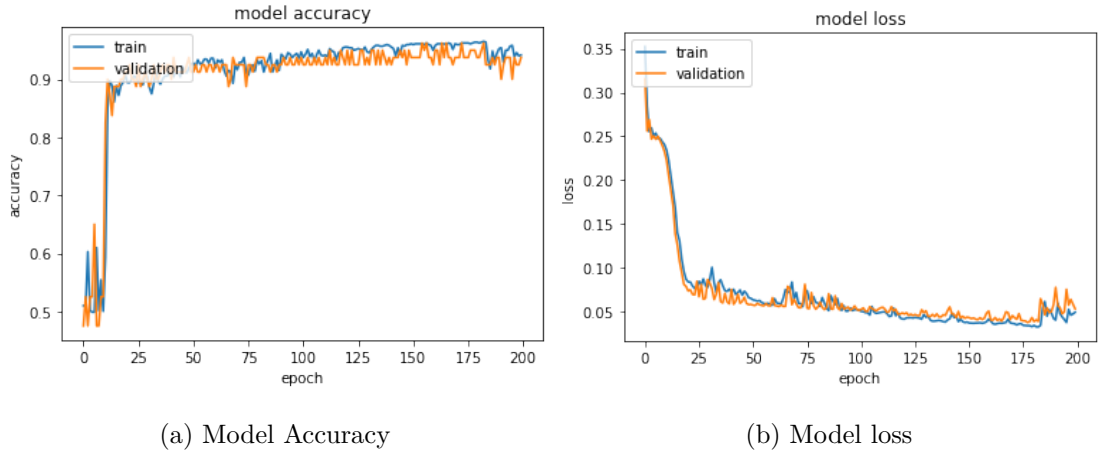


Figure 4.7: VGG16-GRU method graphical representation of loss and accuracy.

The graphs displays the accuracy and loss of the Test and Train datasets while using the fundamental VGG16-GRU model. The y-axis shows the accuracy and loss and the x-axis shows the model training epochs. In this we have attained 93.5 percent accuracy and 5.32 percent loss. These results we have attained by first time running the model.

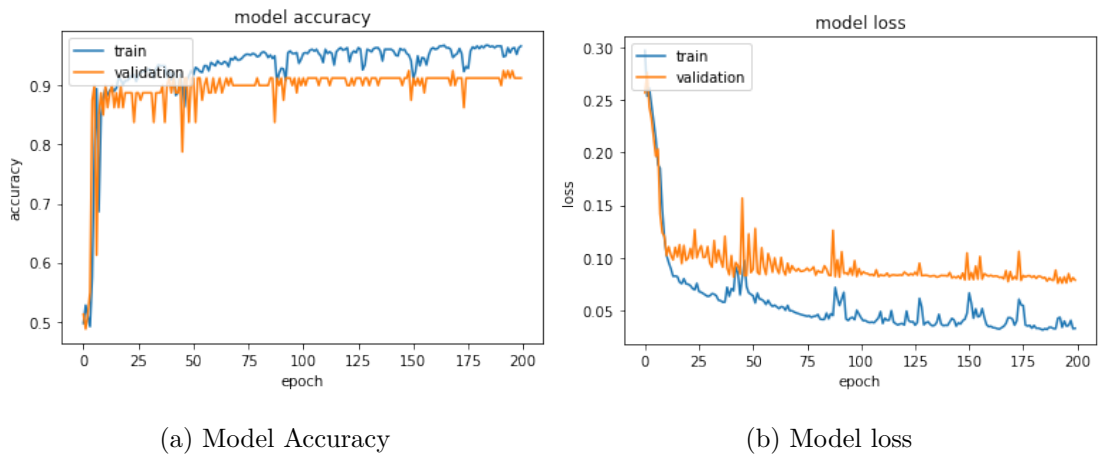
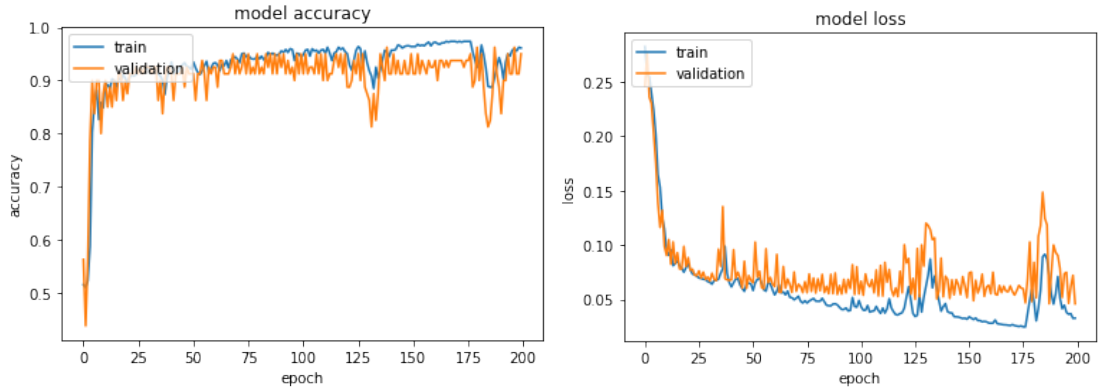


Figure 4.8: VGG16-LSTM method graphical representation of loss and accuracy.

The graphs displays the accuracy and loss of the Test and Train datasets while using the fundamental VGG16-LSTM model. The y-axis shows the accuracy and loss and the x-axis shows the model training epochs. In this we have attained 94 percent accuracy and 4.72 percent loss. These results we have attained by first time running the model.

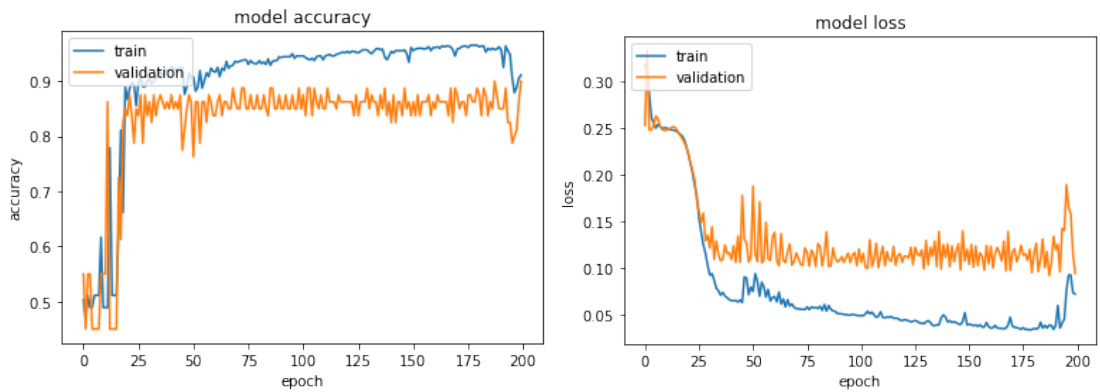


(a) Model Accuracy

(b) Model loss

Figure 4.9: VGG19-LSTM method graphical representation of loss and accuracy.

The graphs displays the accuracy and loss of the Test and Train datasets while using the fundamental VGG19-LSTM model. The y-axis shows the accuracy and loss and the x-axis shows the model training epochs. In this we have attained 91 percent accuracy and 7.32 percent loss. These results we have attained by first time running the model.



(a) Model Accuracy

(b) Model loss

Figure 4.10: VGG19-GRU method graphical representation of loss and accuracy.

The graphs displays the accuracy and loss of the Test and Train datasets while using the fundamental VGG19-GRU model. The y-axis shows the accuracy and loss and the x-axis shows the model training epochs. In this we have attained 92 percent accuracy and 7.32 percent loss. These results we have attained by first time running the model.

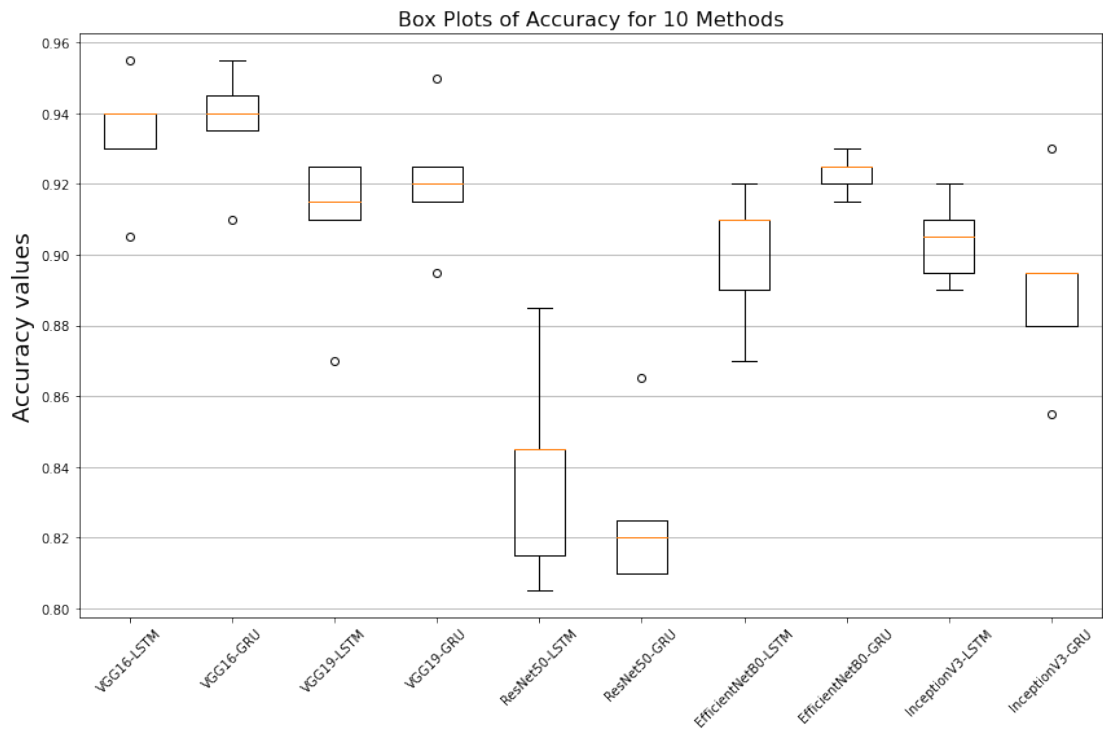


Figure 4.11: This box plot shows the values of accuracy of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the accuracy values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

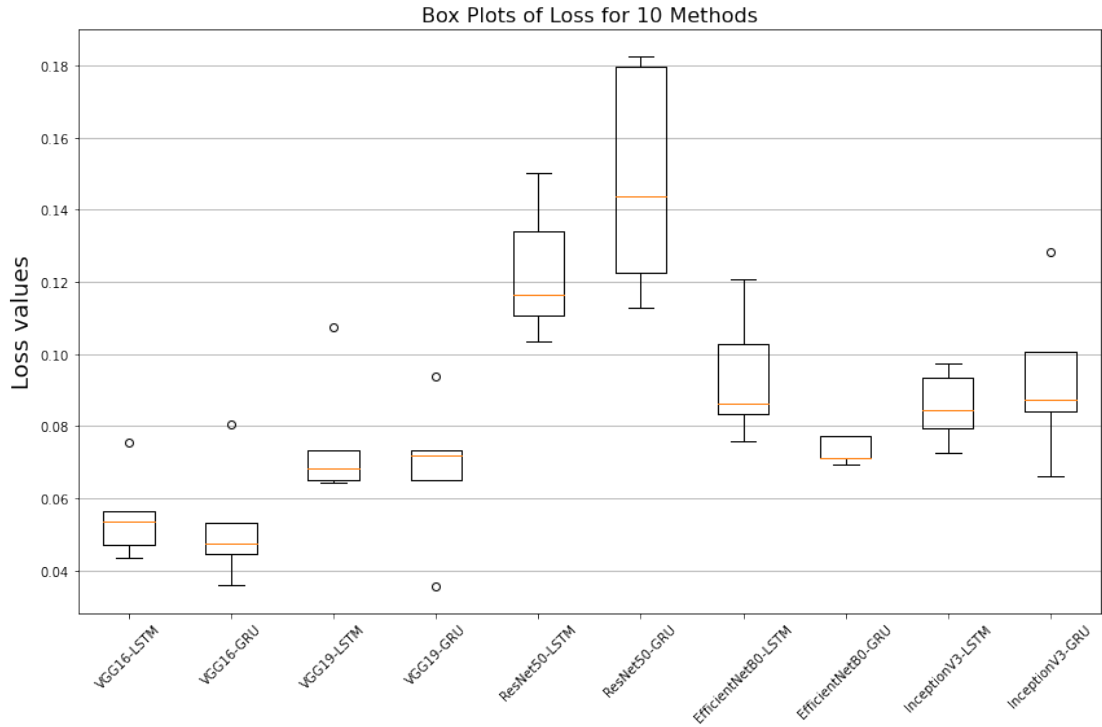


Figure 4.12: This box plot shows the values of loss of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the loss values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

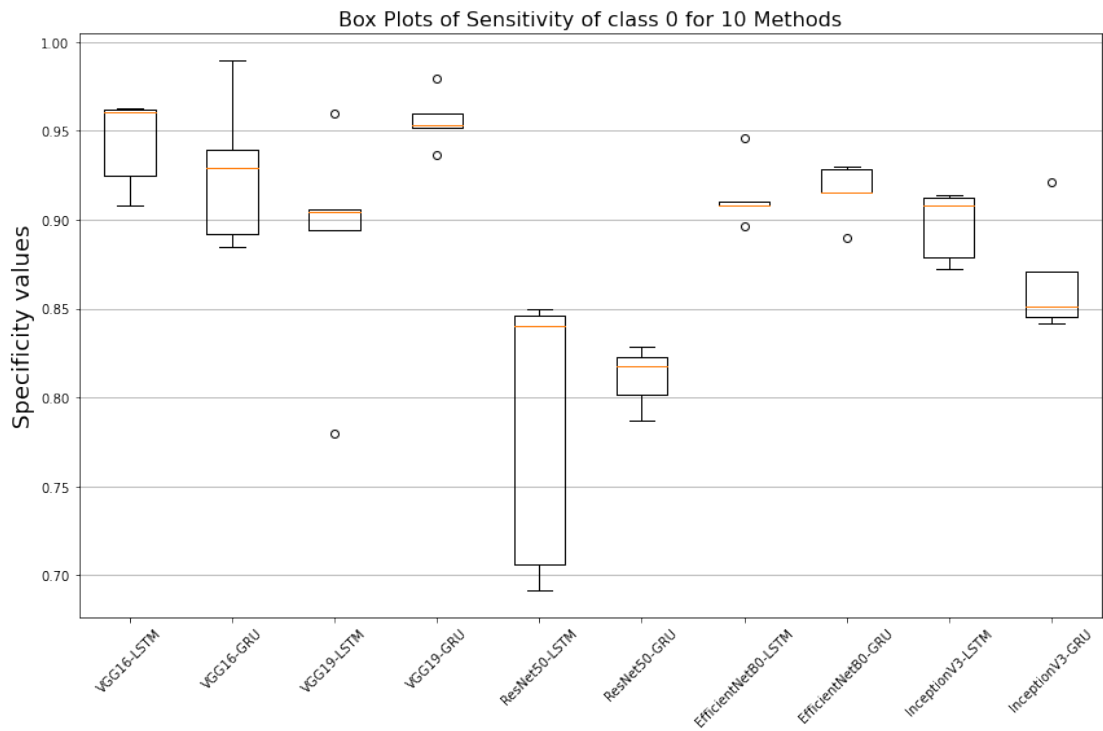


Figure 4.13: This box plot shows the values of sensitivity of class 0 of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the sensitivity of class 0 values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

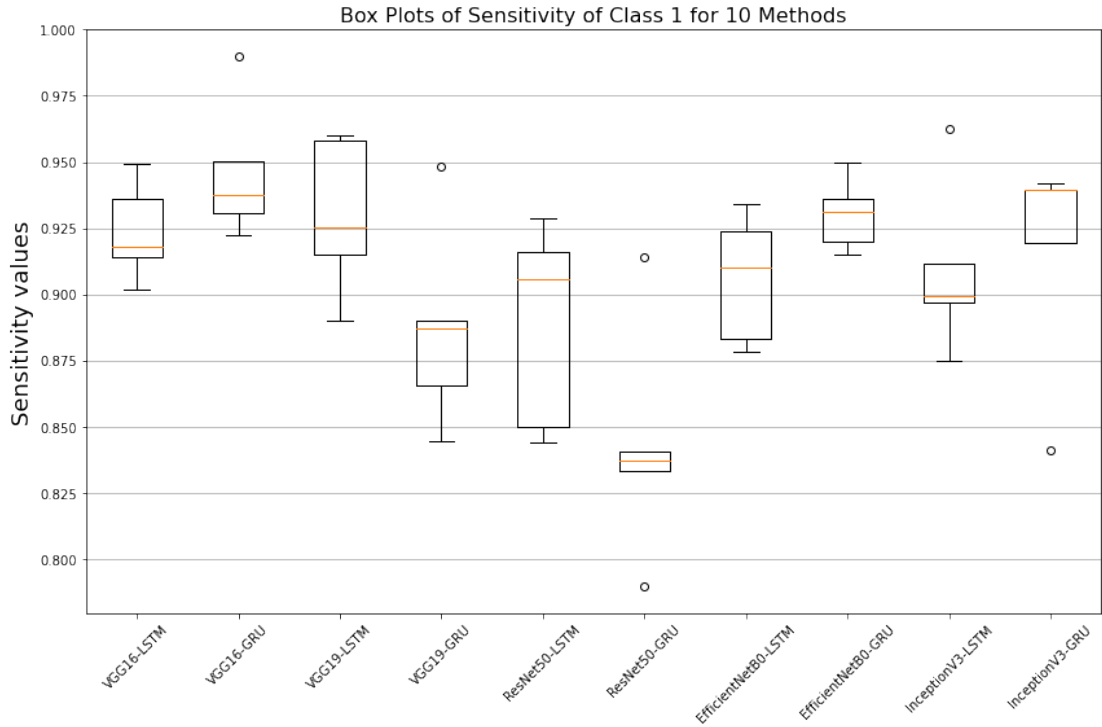


Figure 4.14: This box plot shows the values of sensitivity of class 1 of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the sensitivity of class 1 values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

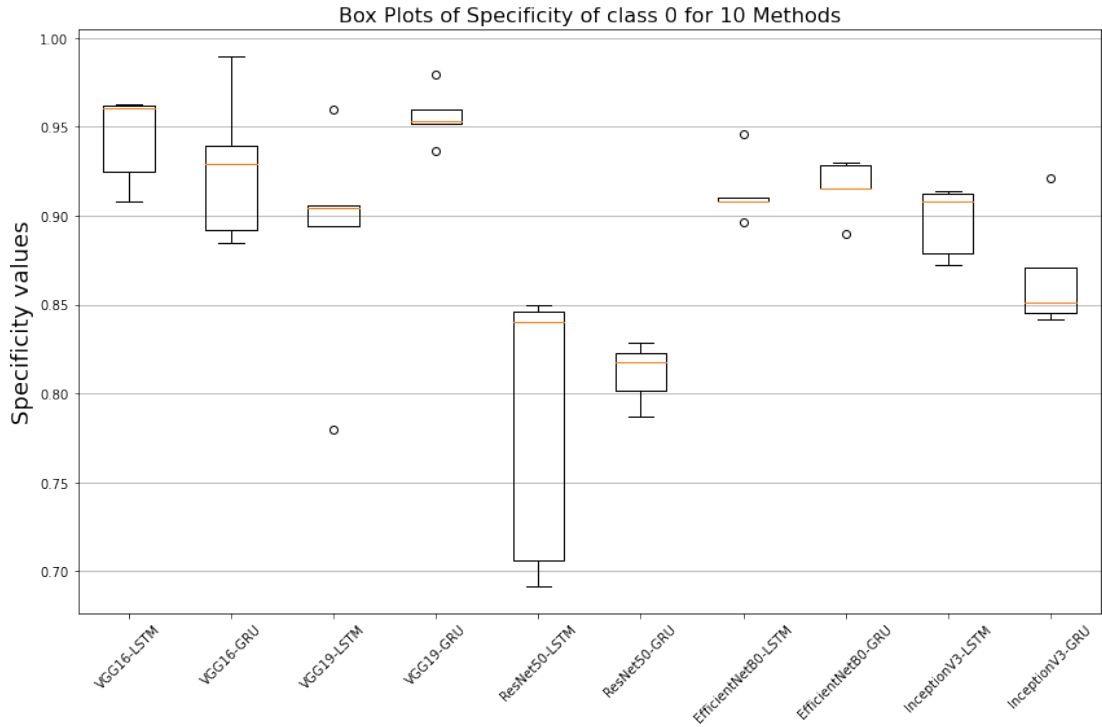


Figure 4.15: This box plot shows the values of specificity of class 0 of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the specificity of class 0 values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

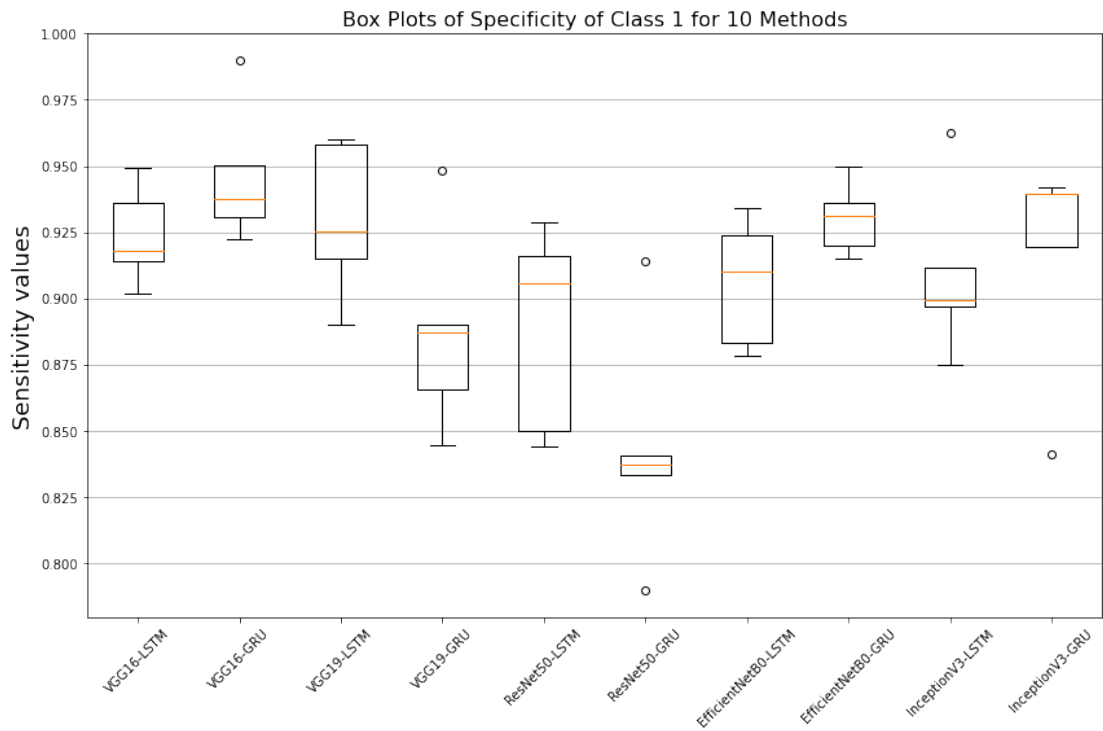


Figure 4.16: This box plot shows the values of specificity of class 1 of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the specificity of class 1 values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

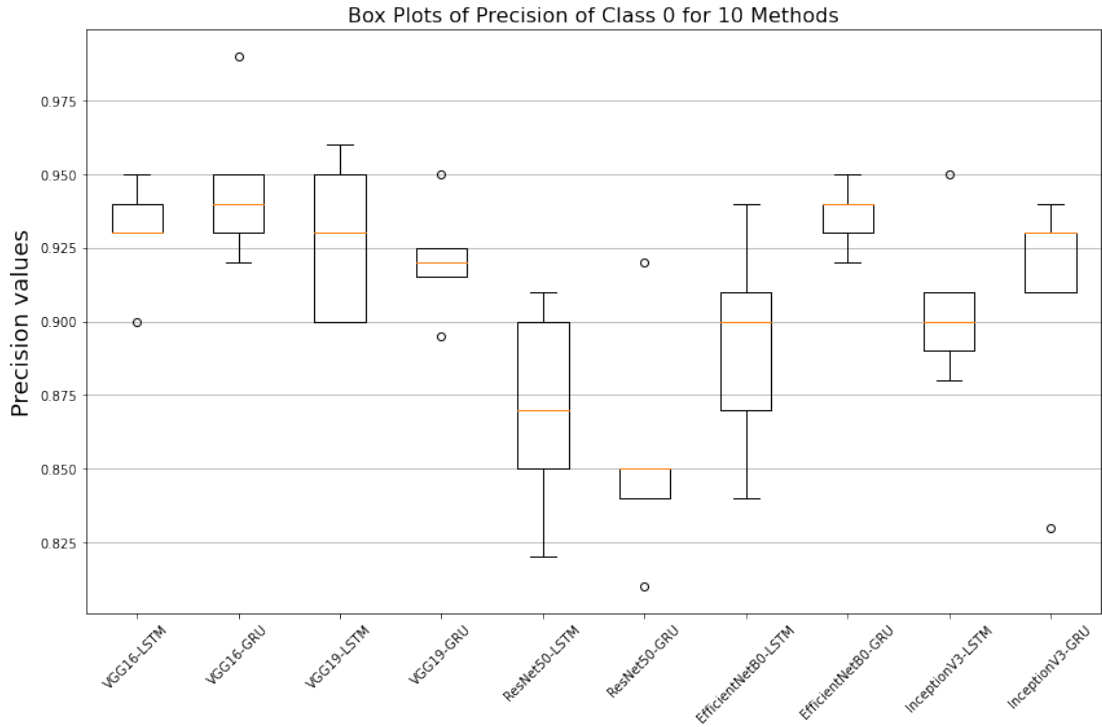


Figure 4.17: This box plot shows the values of precision of class 0 of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the precision of class 0 values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

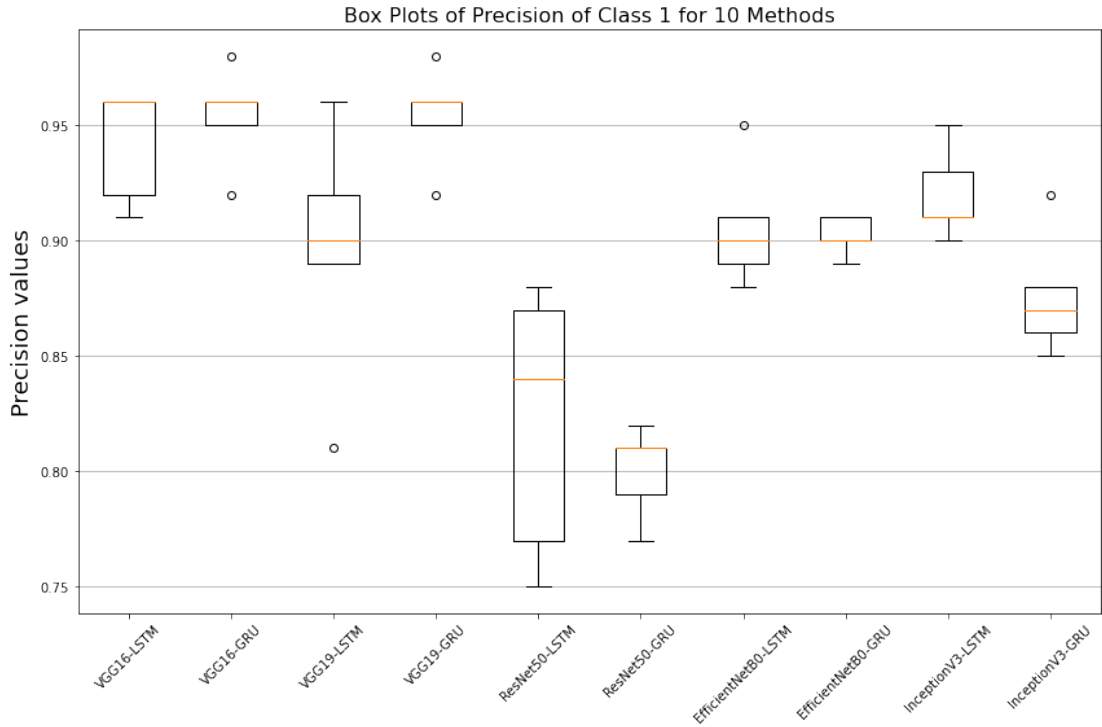


Figure 4.18: This box plot shows the values of precision of class 1 of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the precision of class 1 values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

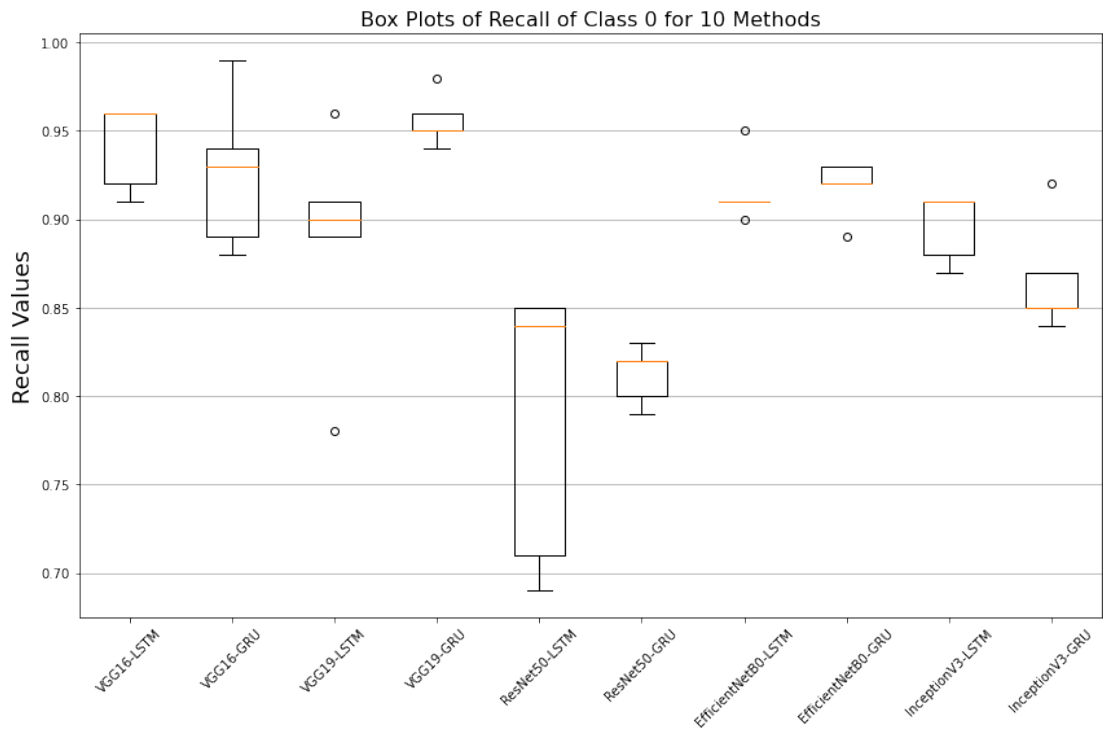


Figure 4.19: This box plot shows the values of recall of class 0 of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the recall of class 0 values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

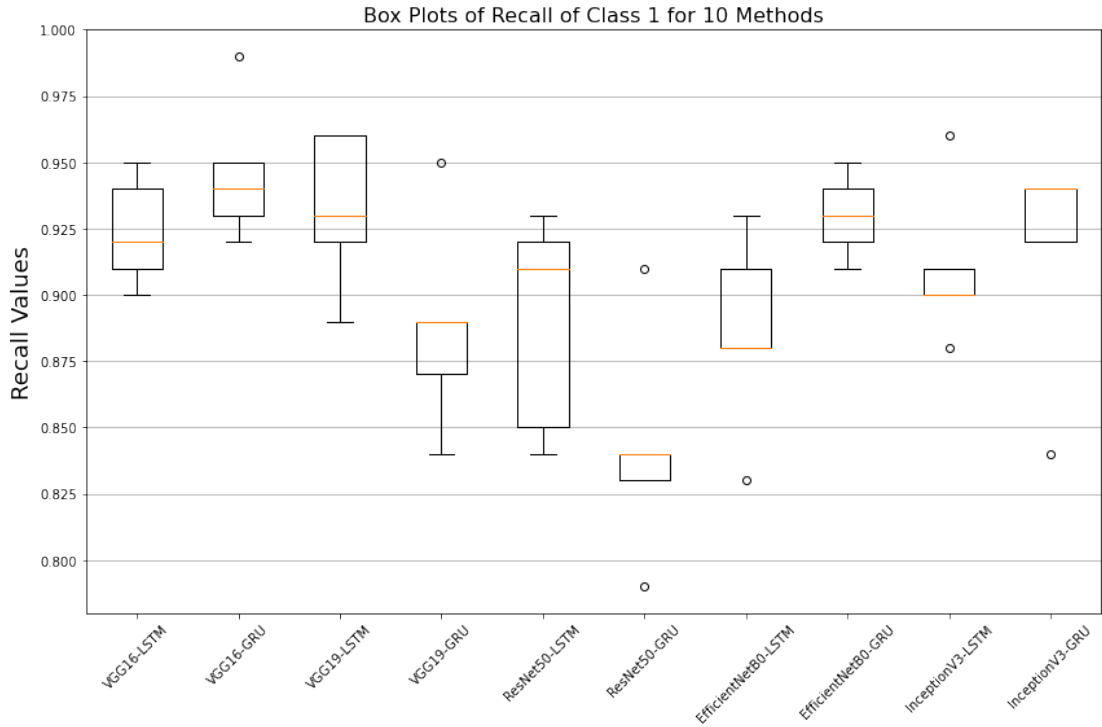


Figure 4.20: This box plot shows the values of recall of class 1 of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the recall of class 1 values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

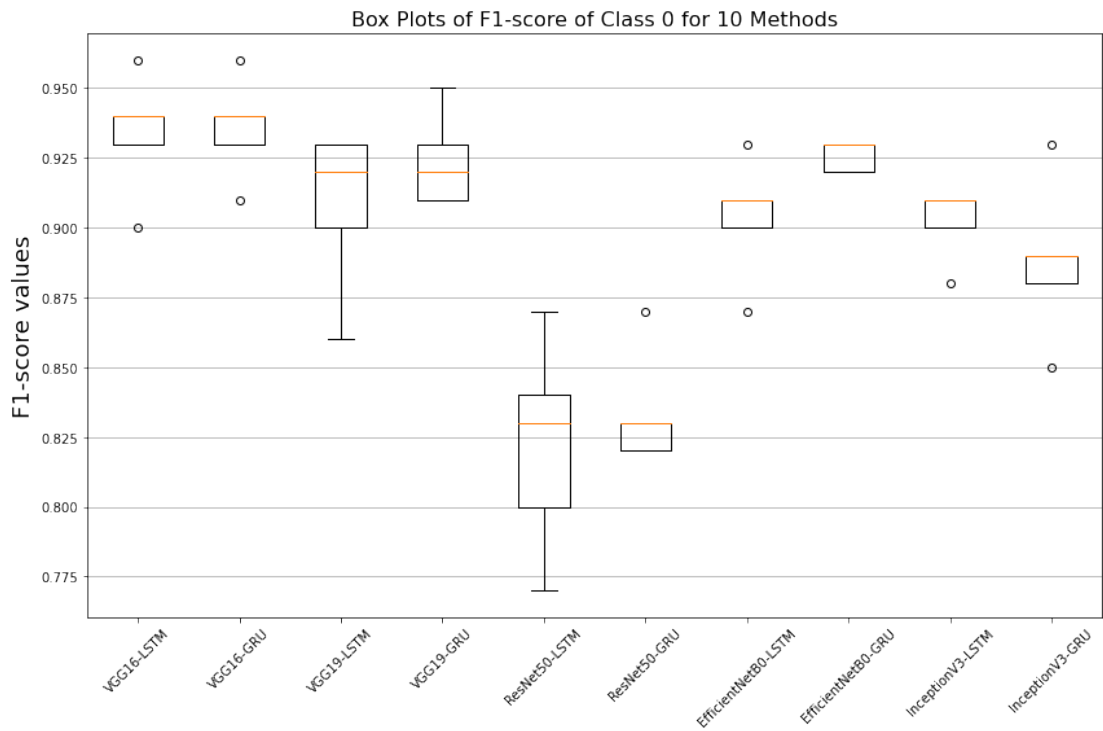


Figure 4.21: This box plot shows the values of f1-score of class 0 of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the f1-score of class 0 values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

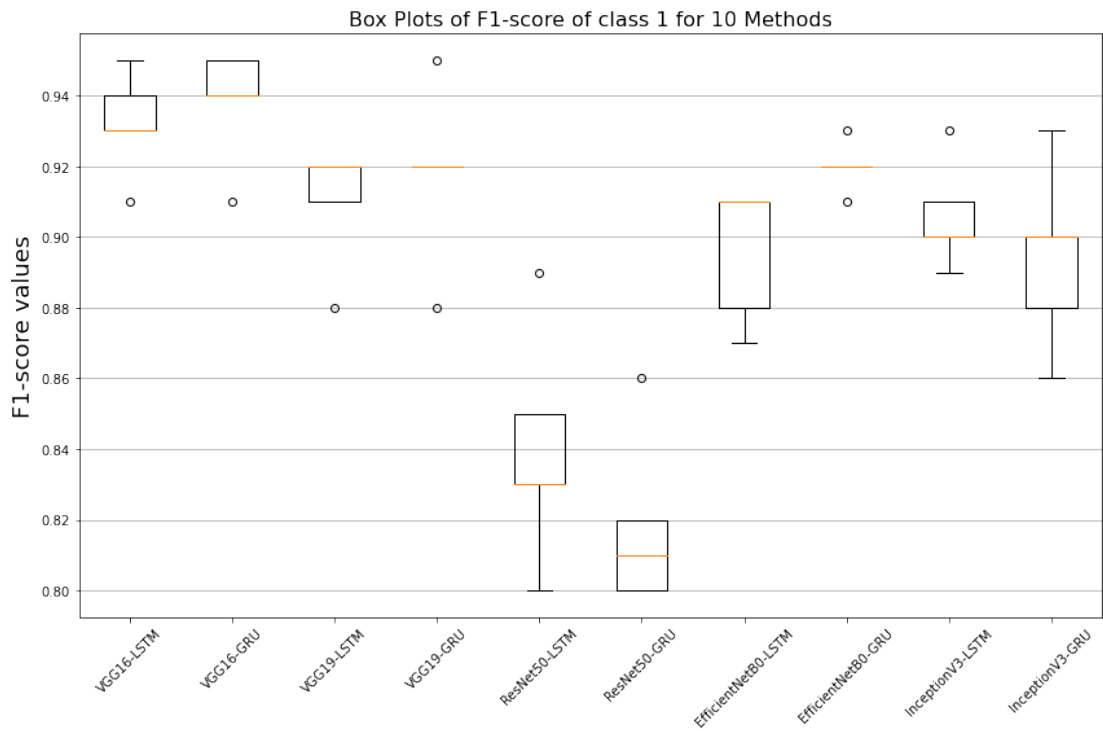


Figure 4.22: This box plot shows the values of f1-score of class 1 of all the methods at its y-axis and the x-axis shows the method names.

The y-axis of this box plot reflects the f1-score of class 1 values of all the techniques, while method names are displayed on the x-axis. Each method's box in box plot indicates the interquartile range (IQR), corresponding to the median 50 percent of the data. The box's bottom border represents the first quartile (Q1), or the twenty-fifth percentile. The box's upper border lines up with the third quarter (Q3), or the seventy-fifth percentile. As a result, the span of the box represents the dispersion of the middle 50 percent of the data. Once the data being analyzed has been organized, a horizontal line within the box depicts the median value, which is the midway number. It divides the data being gathered into two equal halves. Whiskers expand beyond the box to the lowest and highest values within a certain range. The range might differ; however, it is usually fixed to a multiple of the IQR. Values over the whiskers are regarded as outliers. Outside the whiskers, points of data are frequently indicated as distinct points. Outliers are values for the data that deviate dramatically from the remainder of the data and should be investigated further.

Table 4.1: Performance of Neural Networks on Hockey fight videos dataset

methods	loss	accuracy	Precision class 1	Precision class 0	Recall	Recall
					Class 0	Class 1
VGG16-LSTM	0.05526	0.934	0.942	0.93	0.942	0.924
VGG16-GRU	0.05232	0.937	0.928	0.946	0.926	0.946
VGG19-LSTM	0.07574	0.909	0.896	0.928	0.888	0.932
VGG19-GRU	0.06792	0.921	0.954	0.892	0.956	0.888
Resnet50-LSTM	0.123	0.839	0.822	0.87	0.788	0.89
Resnet50-GRU	0.14838	0.826	0.8	0.854	0.812	0.842
EfficientNetB0-LSTM	0.0938	0.90	0.906	0.892	0.916	0.886
EfficientNetB0-GRU	0.07326	0.923	0.912	0.936	0.918	0.93
InceptionV3-LSTM	0.08544	0.904	0.902	0.906	0.896	0.91
InceptionV3-GRU	0.0933	0.891	0.876	0.908	0.866	0.916

Table 4.2: Performance of Neural Networks on Hockey fight videos dataset

methods	F1-score class 0	F1-score class 1	Specificity	Specificity	Sensitivity	Sensitivity
			Class 0	Class 1	Class 0	Class 1
			VGG16-LSTM	0.934	0.932	0.94352
VGG16-GRU	0.936	0.938	0.927	0.9426	0.927	0.9426
VGG19-LSTM	0.908	0.91	0.889	0.92976	0.889	0.92976
VGG19-GRU	0.924	0.918	0.956	0.887	0.956	0.887
Resnet50-LSTM	0.822	0.84	0.7866	0.889	0.7866	0.889
Resnet50-GRU	0.834	0.818	0.81144	0.84282	0.81144	0.84282
EfficientNetB0-LSTM	0.904	0.896	0.91378	0.90589	0.91378	0.90589
EfficientNetB0-GRU	0.926	0.92	0.91576	0.9305	0.91576	0.9305
InceptionV3-LSTM	0.902	0.906	0.89714	0.909	0.89714	0.909
InceptionV3-GRU	0.888	0.894	0.866	0.9162	0.866	0.9162

Table 4.1 and Table 4.2 above represent the results of the research done on the Hockey Fight Videos dataset. For InceptionV3-LSTM we have accuracy values ranging between 89 percent to 92 percent and loss values ranging between 7.26 percent to 9.73 percent. As we have performed five runs to find this range of values we have taken the average of all the values and put them in Table 4.1. For InceptionV3-GRU we have accuracy values ranging between 85.5 percent to 93 percent and loss values ranging between 6.6 percent to 12.82 percent. As we have performed five runs to find this range of values we

have taken the average of all the values and put them in Table 4.1. For EfficientNetB0-LSTM we have accuracy values ranging between 87 percent to 92 percent and loss values ranging between 7.58 percent to 12.09 percent. As we have performed five runs to find this range of values we have taken the average of all the values and put them in Table 4.1. For EfficientNetB0-GRU we have accuracy values ranging between 91.5 percent to 93 percent and loss values ranging between 6.92 percent to 7.74 percent. As we have performed five runs to find this range of values we have taken the average of all the values and put them in Table 4.1. For ResNet50-LSTM we have accuracy values ranging between 80.5 percent to 88.5 percent and loss values ranging between 10.36 percent to 15.03 percent. As we have performed five runs to find this range of values we have taken the average of all the values and put them in Table 4.1. For ResNet50-GRU we have accuracy values ranging between 81 percent to 86.5 percent and loss values ranging between 11.29 percent to 18.26 percent. As we have performed five runs to find this range of values we have taken the average of all the values and put them in Table 4.1. For VGG19-LSTM we have accuracy values ranging between 87 percent to 92.5 percent and loss values ranging between 6.44 percent to 10.76 percent. As we have performed five runs to find this range of values we have taken the average of all the values and put them in Table 4.1. For VGG19-GRU we have accuracy values ranging between 89.5 percent to 95 percent and loss values ranging between 3.55 percent to 9.39 percent. As we have performed five runs to find this range of values we have taken the average of all the values and put them in Table 4.1. For VGG16-LSTM we have accuracy values ranging between 90.5 percent to 95.5 percent and loss values ranging between 4.34 percent to 7.56 percent. As we have performed five runs to find this range of values we have taken the average of all the values and put them in Table 4.1. For VGG16-GRU we have accuracy values ranging between 91 percent to 95.5 percent and loss values ranging between 3.58 percent to 8.06 percent. As we have performed five runs to find this range of values we have taken the average of all the values and put them in Table 4.1. Neural Networks with additional input branches have been employed to analyze data sets and detect the violence with an accuracy range of 82.6 percent-93.7 percent and the other evaluation metrics such as sensitivity, specificity range 78.66 percent-95.6 percent, and 84.28 percent-94.26 percent. On the Hockey fight video dataset, a number of assessment metrics, such as accuracy, sensitivity, specificity, f1-score, a precision are used to assess the model's performance as shown in Table 4.1 and Table 4.2.

Conclusion

The purpose of this work is to detect violence in a video data set using Neural Networks, that has been synthesized. The EfficientNET-BO, Resnet-50, VGG-16, VGG-19 and InceptionV3 are very helpful in extracting the special features. While the LSTM and GRU has done classification for us. The best result that we have achieved is from VGG16-GRU and the worst results that we have achieved is from Resnet50-GRU. Research is required in this field as this can help us in the emerging violence situations. Additionally in future we can have a video surveillance violence detection cameras which give the indication of violence.

Bibliography

- [1] Christian Schuldt, Ivan Laptev, and Barbara Caputo. “Recognizing human actions: a local SVM approach”. In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004*. Vol. 3. IEEE. 2004, pp. 32–36.
- [2] Lena Gorelick et al. “Actions as space-time shapes”. In: *IEEE transactions on pattern analysis and machine intelligence* 29.12 (2007), pp. 2247–2253.
- [3] Ying Wang, Kaiqi Huang, and Tieniu Tan. “Human activity recognition based on r transform”. In: *2007 IEEE conference on computer vision and pattern recognition*. IEEE. 2007, pp. 1–8.
- [4] Moez Baccouche et al. “Action classification in soccer videos with long short-term memory recurrent neural networks”. In: *International Conference on Artificial Neural Networks*. Springer. 2010, pp. 154–159.
- [5] Wanqing Li, Zhengyou Zhang, and Zicheng Liu. “Action recognition based on a bag of 3d points”. In: *2010 IEEE computer society conference on computer vision and pattern recognition-workshops*. IEEE. 2010, pp. 9–14.
- [6] Enrique Bermejo Nievas et al. “Violence detection in video using computer vision techniques”. In: *Computer Analysis of Images and Patterns: 14th International Conference, CAIP 2011, Seville, Spain, August 29-31, 2011, Proceedings, Part II 14*. Springer. 2011, pp. 332–339.
- [7] Tal Hassner, Yossi Itcher, and Orit Kliper-Gross. “Violent flows: Real-time detection of violent crowd behavior”. In: *2012 IEEE computer society conference on computer vision and pattern recognition workshops*. IEEE. 2012, pp. 1–6.
- [8] Shuiwang Ji et al. “3D convolutional neural networks for human action recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2012), pp. 221–231.

BIBLIOGRAPHY

- [9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012).
- [10] Alexander Grushin et al. “Robust human action recognition via long short-term memory”. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2013, pp. 1–8.
- [11] Akihiko Torii et al. “Visual place recognition with repetitive structures”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 883–890.
- [12] Claire-Hélène Demarty et al. “Benchmarking violent scenes detection in movies”. In: *2014 12th international workshop on content-based multimedia indexing (CBMI)*. IEEE. 2014, pp. 1–6.
- [13] Chunhui Ding et al. “Violence detection in video by using 3D convolutional neural networks”. In: *Advances in Visual Computing: 10th International Symposium, ISVC 2014, Las Vegas, NV, USA, December 8-10, 2014, Proceedings, Part II 10*. Springer. 2014, pp. 551–558.
- [14] Karen Simonyan and Andrew Zisserman. “Two-stream convolutional networks for action recognition in videos”. In: *Advances in neural information processing systems* 27 (2014).
- [15] Yong Du, Wei Wang, and Liang Wang. “Hierarchical recurrent neural network for skeleton based action recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1110–1118.
- [16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.
- [17] Paolo Rota et al. “Real-life violent social interaction detection”. In: *2015 IEEE international conference on image processing (ICIP)*. IEEE. 2015, pp. 3456–3460.
- [18] Ismael Serrano Gracia et al. “Fast fight detection”. In: *PLoS one* 10.4 (2015), e0120448.
- [19] Vivek Veeriah, Naifan Zhuang, and Guo-Jun Qi. “Differential recurrent neural networks for action recognition”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4041–4049.

- [20] Pichao Wang et al. “Action recognition from depth maps using deep convolutional neural networks”. In: *IEEE Transactions on Human-Machine Systems* 46.4 (2015), pp. 498–509.
- [21] S Jeba Berlin and Mala John. “Human interaction recognition through deep learning network”. In: *2016 IEEE international Carnahan conference on security technology (ICCST)*. IEEE. 2016, pp. 1–4.
- [22] Guankun Mu, Haibing Cao, and Qin Jin. “Violent scene detection using convolutional neural networks and deep audio features”. In: *Pattern Recognition: 7th Chinese Conference, CCPR 2016, Chengdu, China, November 5-7, 2016, Proceedings, Part II 7*. Springer. 2016, pp. 451–463.
- [23] Zhiwu Huang et al. “Deep learning on lie groups for skeleton-based action recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 6099–6108.
- [24] Zihan Meng, Jiabin Yuan, and Zhen Li. “Trajectory-pooled deep convolutional networks for violence detection in videos”. In: *Computer Vision Systems: 11th International Conference, ICVS 2017, Shenzhen, China, July 10-13, 2017, Revised Selected Papers 11*. Springer. 2017, pp. 437–447.
- [25] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. “Deep learning in bioinformatics”. In: *Briefings in bioinformatics* 18.5 (2017), pp. 851–869.
- [26] Tobias Senst et al. “Crowd violence detection using global motion-compensated lagrangian features and scale-sensitive video-level representation”. In: *IEEE transactions on information forensics and security* 12.12 (2017), pp. 2945–2956.
- [27] Swathikiran Sudhakaran and Oswald Lanz. “Learning to detect violent videos using convolutional long short-term memory”. In: *2017 14th IEEE international conference on advanced video and signal based surveillance (AVSS)*. IEEE. 2017, pp. 1–6.
- [28] Dong Li et al. “Unified spatio-temporal attention networks for action recognition in videos”. In: *IEEE Transactions on Multimedia* 21.2 (2018), pp. 416–428.
- [29] Ismael Serrano et al. “Fight recognition in video using hough forests and 2D convolutional neural network”. In: *IEEE Transactions on Image Processing* 27.10 (2018), pp. 4787–4797.

- [30] Qing Xia et al. “Real time violence detection based on deep spatio-temporal features”. In: *Biometric Recognition: 13th Chinese Conference, CCBR 2018, Urumqi, China, August 11-12, 2018, Proceedings 13*. Springer. 2018, pp. 157–165.
- [31] Ke Xu, Xinghao Jiang, and Tanfeng Sun. “Anomaly detection based on stacked sparse coding with intraframe classification strategy”. In: *IEEE Transactions on Multimedia* 20.5 (2018), pp. 1062–1074.
- [32] Peipei Zhou et al. “Violence detection in surveillance video using low-level features”. In: *PLoS one* 13.10 (2018), e0203668.
- [33] E Fenil et al. “Real time violence detection framework for football stadium comprising of big data analysis and deep learning through bidirectional LSTM”. In: *Computer Networks* 151 (2019), pp. 191–200.
- [34] Wei Song et al. “A novel violent video detection scheme based on modified 3D convolutional neural networks”. In: *IEEE Access* 7 (2019), pp. 39172–39179.
- [35] Fath U Min Ullah et al. “Violence detection using spatiotemporal features with 3D convolutional neural network”. In: *Sensors* 19.11 (2019), p. 2472.
- [36] Qianru Zhang et al. “Recent advances in convolutional neural network acceleration”. In: *Neurocomputing* 323 (2019), pp. 37–51.
- [37] Simone Accattoli et al. “Violence detection in videos by combining 3D convolutional neural networks and support vector machines”. In: *Applied Artificial Intelligence* 34.4 (2020), pp. 329–344.
- [38] Anuja Jana Naik and MT Gopalakrishna. “Deep-violence: individual person violent activity detection in video”. In: *Multimedia Tools and Applications* 80.12 (2021), pp. 18365–18380.
- [39] Fernando J Rendón-Segador et al. “Violencenet: Dense multi-head self-attention with bidirectional convolutional lstm for detecting violence”. In: *Electronics* 10.13 (2021), p. 1601.
- [40] Liang Ye et al. “Campus violence detection based on artificial intelligent interpretation of surveillance video sequences”. In: *Remote Sensing* 13.4 (2021), p. 628.
- [41] Seyed Mehdi Mohtavipour, Mahmoud Saeidi, and Abouzar Arabsorkhi. “A multi-stream CNN for deep violence detection in video sequences using handcrafted features”. In: *The Visual Computer* (2022), pp. 1–16.

APPENDIX A

First Appendix

The separate numbering of appendices is also supported by LaTeX. The *appendix* macro can be used to indicate that following chapters are to be numbered as appendices. Only use the *appendix* macro once for all appendices.